XPRT.

Magazine N°15/2023

Transforming Beliefs: Embracing Growth

Self-Guided Meditations With AI On Azure

Reflections of a DevOpsologist

Let's Playwright with .NET 6 MVC

Understanding the Value of Value Stream Mapping

InnerSource

Xebia | Xpirit

Transforming your business will not work without the right knowledge

Certified Microsoft Azure Fundamentals (AZ-900)

Certified Microsoft Azure Administrator (AZ-104)

Certified Microsoft Azure Developer (AZ-204)

Designing Microsoft Azure Infrastructure Solutions (AZ305)

Certified Microsoft DevOps Engineer Expert (AZ-400)

Learning Journey

Azure DevOps Engineer Expert (AZ-900 • AZ-104 • AZ-400)

Azure DevOps Engineer Expert (AZ-900 • AZ-204 • AZ-400)

Azure Solutions Architect Expert (AZ-900 • AZ-104 • AZ-305)

Azure Developer Associate (AZ-900 • AZ-204)

Azure Administrator Associate (AZ-900 • AZ-104)

View all training options online.

Xebia | Xpirit

Colophon

XPRT. Magazine N° 15/2023

Editorial Office
Xebia | Xpirit Netherlands BV

This magazine was made by Xebia | Xpirit
Erwin Staal, Dennis Thie, Matthijs van der Veer, Olena Borzenko, Patrick de Kruijf, Thijs Limmen, Colin Dembovsky (GitHub), Heidi Araya, Thiago Custodio, Michael Contento, Jasper Gilhuis, Kristof Riebbels, Danny van der Kraan, Arjan van Bekkum,

Contact
Xebia | Xpirit Netherlands BV
Laapersveld 27
1213 VB Hilversum
The Netherlands
Call +31 35 538 19 21
mverhorst@xpirit.com
www.xpirit.com

Layout and Design
Studio OOM /www.studio-oom.nl

Translations
Mickey Gousset (GitHub)

Microsoft Solutions Partner

GitHub Verified Partner

If you prefer the digital version of this magazine, please scan the qr-code.

In this issue of **XPRT.** magazine is about transforming beliefs; embracing growth.

Fall is the season where nature graciously shares its wisdom, shedding old leaves to make way for new growth

René van Osnabrugge

It is October again, and that means a new XPRT magazine. And as this quote nicely describes, this magazine is again full of new knowledge that allows further growth. The quote, generated by ChatGPT is a good example of how our daily lives are already impacted by AI. At Xebia | Xpirit, we use AI daily to be more creative and productive. But as with everything, great power comes with great responsibility. There are many risks, and we strive for a responsible use of AI. But, to be very honest, we think it is awesome. To show how AI can help us be creative, Matthijs van der Veer wrote an article about how AI helps him create a fit-for-purpose meditation every day. But of course, there is more than AI, and in this magazine, we show that there is also a lot of other content to share.

In this very international magazine, with articles from Xebia | Xpirit US, Xebia | Xpirit Belgium, Xebia | Xpirit Germany, Xebia | Xpirit Netherlands, and a guest writer, we again share a variety of knowledge that allows you to grow. As usual, we use our engineering culture pillars to logically group the articles. State of the Art Software Engineering is and always has been an important topic within Xebia | Xpirit. And in this magazine, we share again a broad scope of articles. Although testing might have moved to the background, it is more important than ever. You will find two articles that revolve around testing. Kristoff writes about "Testing with Playwright", a UI Testing framework and Michael wrote a follow-up on his mutation testing article (#magazine 14) about Fuzzing, meaning that you supply a program with invalid, random, or unexpected input until it encounters a crash.

Danny dives into yet another "Ops". GreenOps. GreenOps is all about sustainable software engineering, and we will show you how you can get started with that. And because you might want to start building this out within your own company, Jasper and Arjan wrote an article about InnerSource. So check this out and try some of the suggestions depicted.

Then, in our Smooth Delivery pillar, Erwin talks about Azure Deployment Environments. A new offering that allows dev teams to spin up infrastructure in the cloud easily. In the Power through Platform pillar, Patrick talks about how Azure Policy can help you to be more compliant, and Olena writes about securing the Azure Service Bus.

We also have some articles for you that are less technical but equally valuable. Heidi writes about the value of value stream mapping, enabling you to clarify your digital vision.

We are also proud of our guest writer, Colin Dembovski, who shares his DevOpsologist journey. A journey from South-Africa to the US, where he now works for GitHub. From the more practical side of Sharing knowledge, Thiago shares some secrets to ace the CKAD (Kubernetes) exam.

Last but not least, Dennis and Thijs wrote an article about building the Techorama booth. The booth that you can visit if you are there. A true IoT-enabled Escape room. For me, this shows the innovation power and technical expertise and is a perfect example of how an Epic Workplace can help foster and nurture this.

I think that we again succeeded in creating a diverse magazine, with articles that cover a wide variety of knowledge and that again underpin our mission. Being an authority in the field! Enjoy! </>

Efficient and Secure Software Delivery with Azure Deployment Environments

In March this year, Microsoft made another offering in Azure generally available: Azure Deployment Environments. Azure Deployment Environments lets development teams quickly and easily spin up app infrastructure. This infrastructure is defined in project-based templates that establish consistency and best practices while maximizing security. The infrastructure can be written by, for example, the platform team. A development team can then, on-demand, create secure environments through a self-service experience that accelerates all stages of the software development lifecycle. Azure Deployment Environments are part of Azure Dev Center, which also houses the Azure Dev Boxes.

Author Erwin Staal

If you want to get your hands dirty right away, then use this great tutorial on Microsoft Learn[1]. Want to know a bit more about the service before using it? Read on! Before we dive into more detail on Azure Deployment Environments, let's first look at the problems it tries to solve.

A central approach to managing the cloud

Modern cloud-native applications leverage a lot of different services in the cloud. Managing this infrastructure can be a challenge as it quickly becomes complex. To create secure and compliant environments, one must know a lot about scale, identity, networking, and costs. Quite often, developers are not experts in these areas and, maybe more important, they don't want to be. They want to write the logic that brings value to the business, not build the infrastructure. That means that the required knowledge to build the infrastructure is unavailable in each DevOps team.

Missing required knowledge is one of the reasons organizations tend to have a central team in control of their cloud. Resources are requested through a central IT team. Due to paramount security and compliance concerns, enterprises commonly withhold direct developer access to public cloud platforms like Azure. Many organizations deal with sensitive data, such as personal information or proprietary business data, which necessitates stringent security measures. Allowing developers unmediated access to public cloud services could inadvertently expose critical data or result in non-compliance with industry regulations. This approach prioritizes safeguarding sensitive information and maintaining adherence to established security standards.

The dynamic and scalable nature of public cloud services brings the challenge of cost management to the forefront. Enterprises adopt a centralized model for cloud resource allocation to mitigate potential financial risks. Public cloud platforms operate on a pay-as-you-go basis, making it imperative to control resource provisioning. Enterprises tend to think a central team is better equipped to monitor, track, and optimize resource usage, effectively preventing unforeseen costs resulting from unmanaged or unnecessary resources. They believe this approach contributes to a more predictable and controlled financial landscape.

[1] https://learn.microsoft.com/en-us/azure/deployment-environments/tutorial-deploy-environments-in-cicd-github

Another reason for this central approach is that centralized control over public cloud resource provisioning leads to better-optimized resource allocation and utilization. Without this bird's-eye view, developers might independently create redundant or underutilized resources, leading to inefficiencies and wasted capacity. A central team can assess the organization's overall resource needs, ensure alignment with business objectives, and allocate resources to maximize efficiency and minimize redundancy. This conserves resources and promotes more effective use of cloud infrastructure.

A fundamental challenge in large enterprises is maintaining consistency and collaboration across diverse projects and development teams. A central team-managed approach fosters standardization and collaboration by establishing uniform practices, templates, and configurations for cloud resources. This ensures that all projects adhere to established best practices and configurations, reducing the risk of security vulnerabilities or operational discrepancies arising from misconfigurations. This approach can streamline development efforts, facilitate cross-team collaboration, and contribute to higher-quality outcomes.

Why is this central approach a problem?

While this central approach to managing the cloud environment might seem reasonable at first, as it seems to safeguard critical aspects of enterprise operations, it also introduces challenges. Challenges that Azure Deployment Environment tries to help you solve. One of the most noticeable disadvantages is the potential for slower resource allocation and flexibility. This model forces developers to wait for the central team to allocate the necessary cloud resources, introducing delays in project timelines and inhibiting the agility and responsiveness required in today's fast-paced development landscape. When a central team serves as the gatekeeper for all resource provisioning, there's a risk of becoming a bottle-neck during peak demand periods. Miscommunications or a lack of nuanced understanding between developers and the central team can lead to mismatches in resource allocations, resulting in projects being allocated insufficient or surplus resources.

How can Azure Deployment Environments help?
Striking a balance between central control and developer empowerment remains pivotal for effectively managing public cloud resources within enterprise environments. Azure Deployment Environments is a new service that addresses these challenges by providing developers with a self-service, on-demand environment provisioning experience. A developer portal, the Azure CLI, or CI/CD pipelines can be used to create, delete or redeploy environments. This allows developers to have their environments ready when they need them. In the future, new functionality will be added to the product to set an expiry date on an environment. An environment is then automatically deleted when it expires to prevent the environment from burning money while no longer being used. Another future option that will be added will allow us to automatically scale down the environment during, for example, the weekend. These features will help take control over costs.

Azure Deployment Environments can also reduce the often redundant work a central platform team does. It allows them to configure built-in governance and have centralized control over the environments. The platform engineer would start by creating a Dev Center and a project. A single project typically represents a single development team. Platform engineers can then define the different types of environments a specific project can use. For a specific environment type, they can control who can create that type of environment. For example, a development team can be allowed to create only a development or test type of environment. A quality engineer might be allowed to create the test environments. Finally, a CI/CD pipeline can be configured to create a staging and production environment.

Environments are defined in templates using infrastructure as code practices. This allows for centralized control over resource allocation and management by, for example, a platform team. A Git repository can be attached as a catalog to the service. The service will automatically scan through that repository, identify these infrastructure as code templates, and make them available for developers. While doing so, they will be asked to provide some basic information about the environment. They won't be asked about, for example, the subscription, resource group, or any other Azure governance-related aspect of the environment. That information was already configured by the platform engineer in Azure Deployment Environments, making deploying the environments easier. It also means that any policies applied to the subscriptions or a higher management group will automatically be enforced in any new environment. This helps to keep all environments compliant.

Platform engineers can also configure the identities that will be used to create the environment. Whenever a developer selects to create an environment, the service uses managed identities to perform deployment on behalf of the user. This is more secure and isolated because these managed identities are specific for this environment type and for this development team. A platform engineer can also configure what set of permissions should be assigned to the developers when it is created. Being able to set permissions this granular fits nicely in a zero-trust architecture.

Finally, tags can automatically be applied to all the resources that the developers are creating. In that way, you can continue to use other tools that you might be using to, for example, track and manage the costs of the resources in Azure.

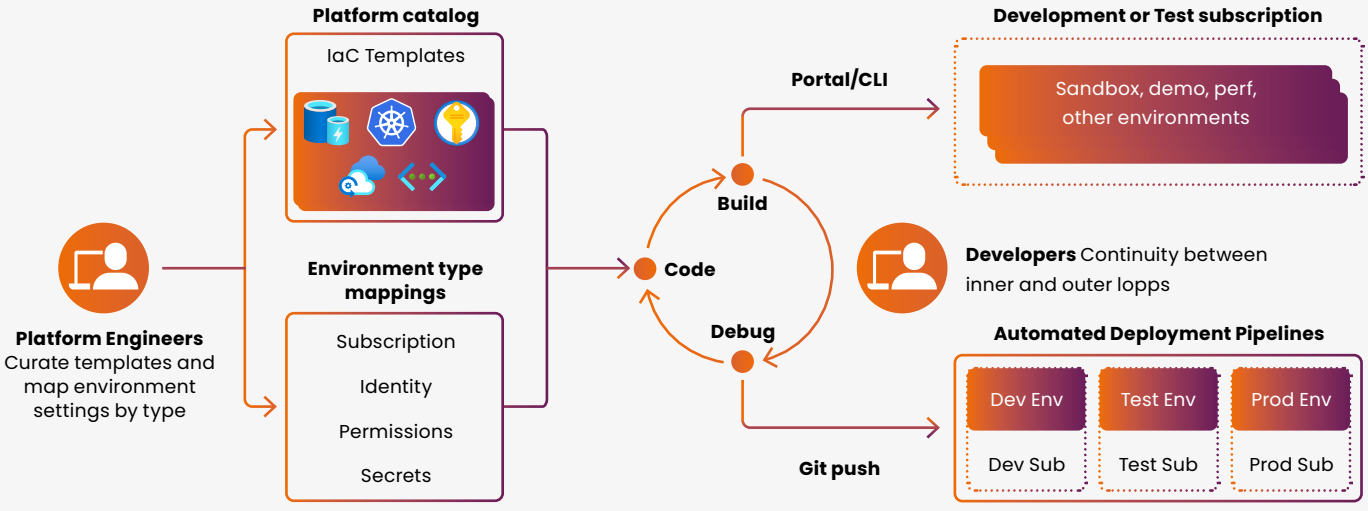


Figure 1: Azure Deployment Environments

We have now seen how Azure Deployment Environment tries to help both the developers and the platform engineers. Now that we know more about the product and how it can help both roles in their work, let's see it in action!

How would I use Azure Deployment Environments as a Platform Engineer?

As the introduction mentions, Azure Deployment Environments are part of Azure Dev Center. This service also houses the Azure Dev Boxes. Azure Dev Boxes are pre-configured development environments that developers can use to start developing applications quickly. In this article, we won't go into detail on Azure Dev Boxes[2], but you can read more about them.

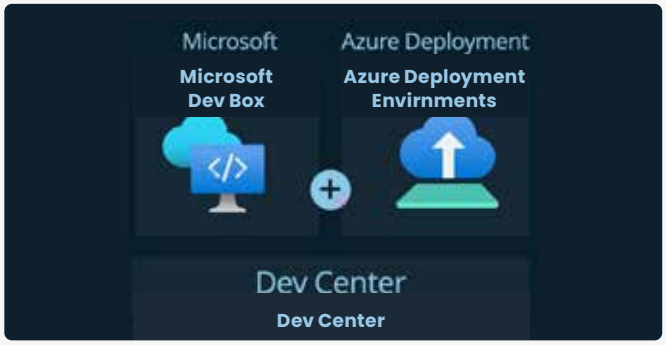


Figure 2: Dev Center

After creating a Dev Center, you need to configure four items to get started with Deployment Environments:

- A Project
- Environment Types
- A Template Catalog
- An identity

Creating a Project

Projects allow you to manage environments and Dev Boxes on a team level. Creating a project is very straightforward. The basic configuration requires a name and the resource group where you want it deployed.

Creating Environment Types

Environment types help define the environments that development teams can create. These are later referenced from within a project, and you can then provide unique deployment settings for each type. Examples here could be development, test, and production. Creating an environment type on the Dev Center level only requires a name and allows you to add default tags. These tags are added to all resources created in an environment of this type. We will see shortly what we can do with these environment types within a project.

Configuring the identity

The next thing that needs to be configured is the identity. This identity is used to deploy the environments and needs to have the proper permissions to do so. The identity can be a managed identity or a service principal. We will later see that a more granular approach is possible where you can configure different identities for different projects and different environment types.

Creating a Template Catalog

When provisioning an environment, it is created using a template definition. A template definition is a set of Terraform (in preview) or ARM files that define the infrastructure to be deployed. A template catalog is a collection of these template definitions. A template catalog can be created on the Dev Center level and referenced from within a project. This gives you a central place to manage all your template definitions. The following image shows you how to create a new template catalog.

Figure 3: Template Definition

A template catalog is a reference to a GIT repository that contains the templates. As you can see in the image, you can specify the URI, the branch, and the path within the repo that holds your definitions. A PAT is used to access the GIT repository and must be stored in a Key Vault. The identity configured in the dev center needs access to that Key Vault. We will see how to create a template definition in a bit.

Now that we have configured the Dev Center, we can start configuring the new project we just created.

Configuring the Project

Remember we talked about environment types earlier? We can now use those environment types to configure the project. The following image shows the creation of a new environment type within the project.

[2] https://docs.microsoft.com/en-us/azure/dev-center/dev-boxes/overview

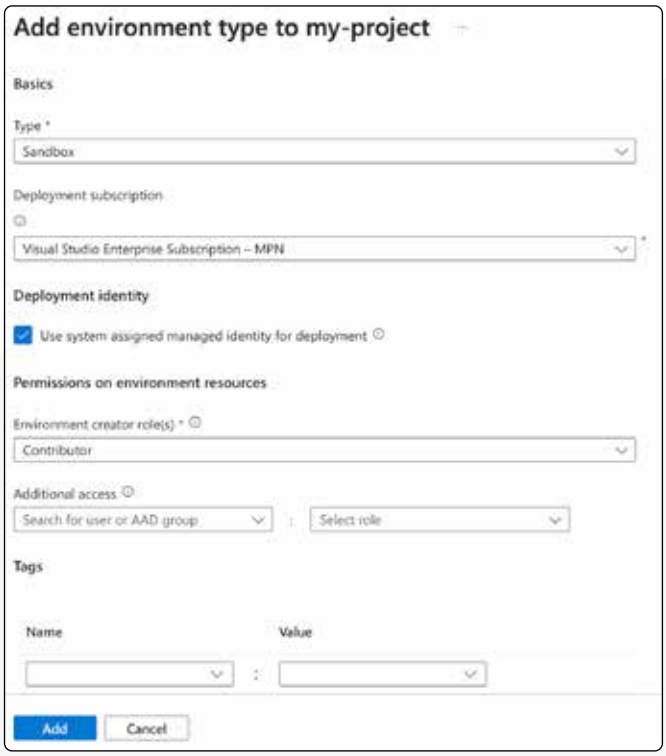


Figure 4: Environment Settings

In the type field, we can select any types created on the Dev Center level that weren't used yet. Next, we can choose the subscription used to deploy the environment. I'm using only one subscription in multiple environment types in the sample. In a real-world scenario, having a separate subscription for test and production workloads per team would be advised.

Next, we can configure a few options around identities and permissions. First, we can select the identity used to deploy environments of this type. This allows you to have different identities for different environment types and ensures we never have a single identity with access to all environments. Then we can configure the permissions assigned to new environments of this type to whoever creates it. When you create an environment using the Azure CLI, the permissions are assigned to you. If you create the environment in a CI/CD pipeline, the identity that executes the pipeline will receive the permissions. Below that option, you get to specify additional users or groups that need specific permissions on the newly created environment. You could, for example, assign read permissions to a team when they do not already have those permissions on the subscription level.

The following image shows the permissions set when creating a new environment with the settings shown in the image above.

Figure 5: Environment Role Assignments

We can see that 'my-project-Test', my GitHub Actions user for this environment type, was assigned the Contributor role. The 'my-project/environmentTypes/Test' identity, the system-assigned managed identity for this specific environment type, gets the Owner permissions.

The last configuration you want to do on a project is to configure who can use it. Who created the project will automatically be an administrator and have the 'DevCenter Project Admin' role assigned. Next to that, you can assign the 'Deployment Environments User' role to the team that belongs to this project. They will then be able to create and use the environments that belong to this project.

Now that we've configured the dev center and the project, it is time to create a template definition that we can deploy!

Creating a Template Definition

The following image shows the 'Environments' folder used when configuring the catalog in the dev center. The folder structure is important as it determines which template definitions are available when creating a new environment. Each folder here represents a single template.

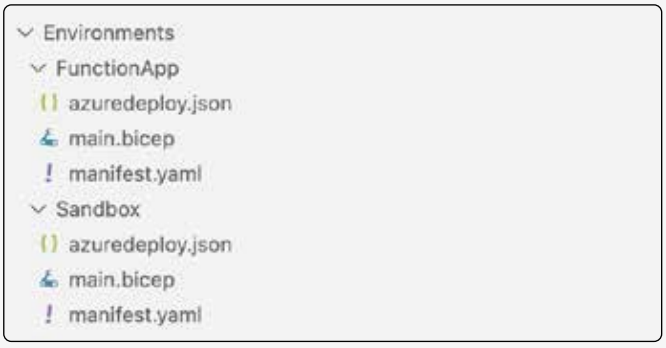


Figure 6: Catalog Folder Structure

Each template definition needs to have a 'manifest.yaml'. This 'manifest.yaml' file contains the metadata for the template definition. This information will be used to, for example, populate the UI, as we will see later on when creating an environment. Here's the 'manifest.yaml' for the 'FunctionApp' template definition.

```
name: FunctionApp
version: 1.0.0
summary: Azure Function App Environment
description: Deploys an Azure Function App,
Storage Account, and Application Insights
runner: ARM
templatePath: azuredeploy.json

parameters:
  - id: name
    name: Name
    description: 'Name of the Function App.'
    type: string
    required: true

  - id: supportsHttpsTrafficOnly
    name: 'Supports Https Traffic Only'
    description: 'Allows HTTPS traffic only to
    Storage Account and Functions App if set to true.'
    type: boolean

  - id: runtime
    name: Runtime
    description: 'The language worker runtime to load
    in the function app.'
    type: string
    allowed:
      - node
      - dotnet
      - java
    default: dotnet
```

This file first contains standard fields like a name, version, and description. The runner specifies whether your templates are written using ARM or Terraform. The use of Terraform is, at the time of writing this article, still in preview, but you can sign up[3] to use it. In this example, an ARM template is used hence the runner: ARM.
The templatePath points to the ARM template used to deploy the environment. Bicep[4] is not supported but, since it's the successor of ARM templates, is preferred over using an ARM template.

Luckily you can use Bicep and transpile that into an ARM template, as is done in this example. The `main.bicep` contains all the required resources for the Function App to work and can then be converted into `azuredeploy.json`' using the following command:

```
az bicep build --file main.bicep --outfile
azuredeploy.json
```

The parameters section defines the parameters that are required to deploy the template. These parameters are used to populate the form when creating a new environment.

With that in place, we can now create a new environment!

How do I use Azure Deployment Environments as a developer?
There are multiple ways a developer could use Deployment Environments. The first of them is manually deploying a new environment. That is done using the Dev Portal or a dev tool like the Azure CLI. Using your CI/CD pipeline to deploy a new environment is the other way.

Figure 7: Developer Experience

Manually deploying a new environment
Let's start with having a look at the Dev Portal. The Dev Portal is a web-based portal that can be used to manage Azure Deployment Environments. It is found at https://devportal.microsoft.com. It provides an overview of all the currently deployed environments and allows a developer to deploy a new one. The exact steps to create a new environment depend on whether you also have access to Dev Boxes. If you don't, you will see a blue button on the top-left corner saying 'New Environment'. If you have access to Dev Boxes, that button will be a drop-down, and one of the two options will say 'New Environment'.

[3] https://5a3318f6fcc34e41bf99d46845944055.svc.dynamics.com/t/formsandbox/gah7wEnZR-zwvk2WzWsrrqj5FETpKfjhu-DGuUAkgw0/7e15cc8f-e4de-ed11-8847-6045bd023ad4?ad=
[3] https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/overview?tabs=bicep

Clicking that 'New Environment'-button will show you the following form:

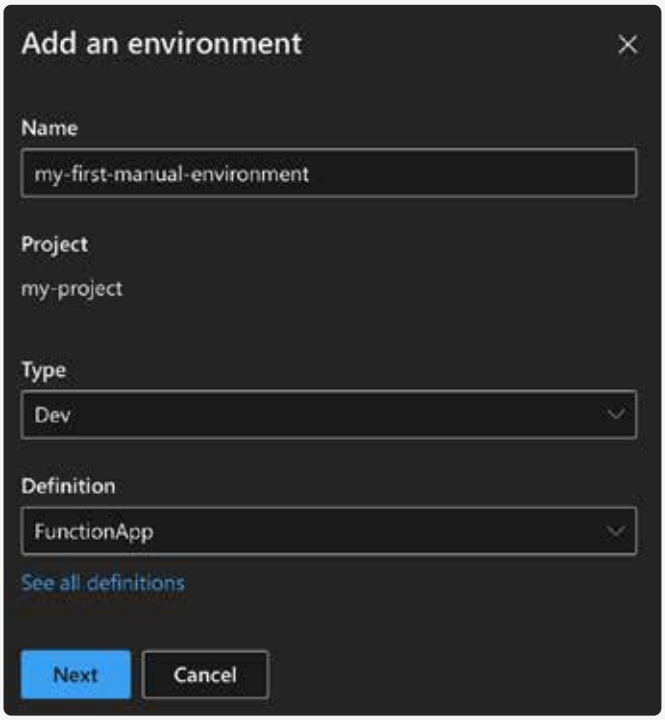


Figure 8: Create New Environment

Here, one must enter a name for the new environment, select the environment type like 'Dev', and select the template definition. We will later talk about how these template definitions are created. After clicking next, a few additional parameters required on this specific template definition must be inserted.

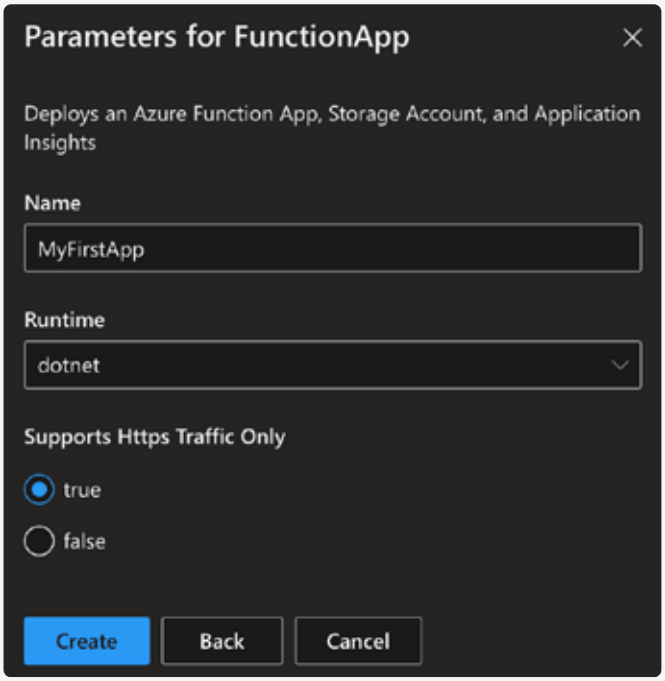


Figure 9: Create New Environment Parameters

Once the form is submitted, the environment is deployed. This can take a few minutes. Once the environment is deployed, it will appear in the environment list.

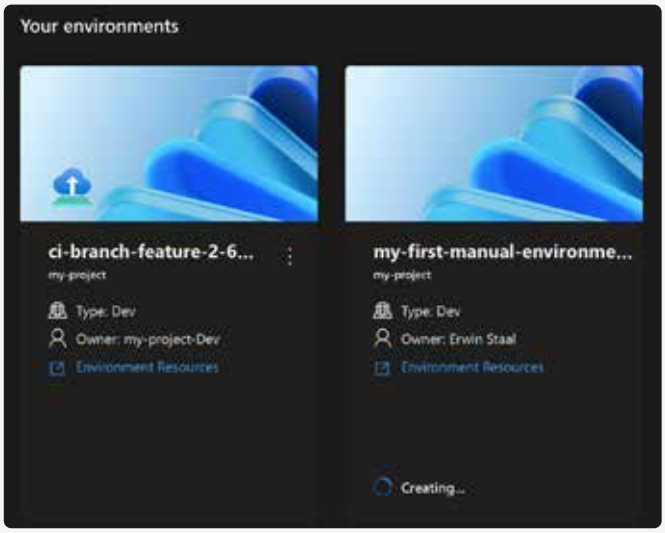


Figure 10: Environment List

That same environment can also be created using the Azure CLI. The Azure CLI command is 'az devcenter dev environment create'. The following command creates a new environment called 'my-dev-environment' using the template definition 'my-template-definition'. One also needs to specify in which project and dev center the environment should be created. The catalog name is the name of the catalog that contains the template definition.

```
az devcenter dev environment create \
   --name 'my-dev-environment' --environment-type 'Dev' \
   --dev-center ${{ vars.AZURE_DEVCENTER }}
   --project ${{ vars.AZURE_PROJECT }} \
   --catalog-name ${{ vars.AZURE_CATALOG }}
   --environment-definition-name 'my-template-definition'
```

Deploying a new environment using CI/CD

Another interesting use of the Azure Deployment Environments is to use it in your CI/CD pipeline. This allows you to create a new environment for every branch or pull request that is being created. For you, as the one who created the branch or PR, it allows you to test your pull request in a real, completely isolated environment. Those who need to review your PR or test it can do so without having to deploy anything themselves.

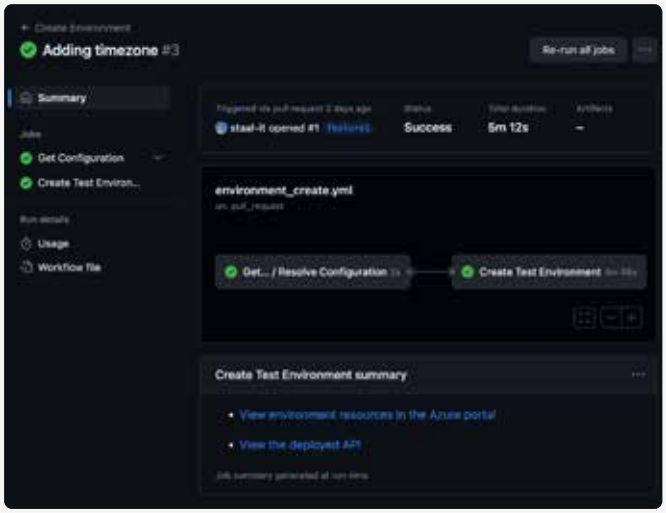


Figure 11: PR Create Environment

In the image above, we see that a link to the deployed environment in Azure and a link to the deployed API on the Azure Function are shown in the summary of a GitHub Action. That same information can also be added as a comment to the PR, as shown below.

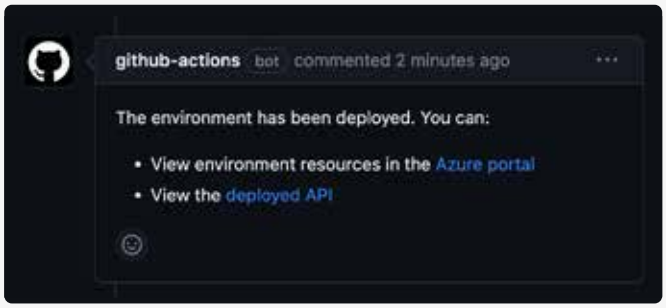


Figure 12: PR Comment

This allows the reviewer to test the API and see if the changes work as expected.

In short, these are the steps in the GitHub Action:

- Build a .NET Core API (a very simple API that has a single endpoint that returns timezone information)
- Log in to Azure using the Azure CLI
- Create the environment using the Azure CLI
- Deploy the API to the environment using the Azure CLI
- Add a comment to the PR with the link to the deployed API and environment

These sample GitHub Actions can be found in a repository shared by Microsoft, as used in the tutorial mentioned in the introduction. My slightly modified version can be found here (https://github.com/staal-it/deployment-environments-demo).

You build it, you run it?

Microsoft lists one of the benefits of this new tool: platform teams can manage the infrastructure by authoring the template definitions. Teams can then use the self-service capabilities of the tools to use them. From a compliance and governance standpoint, this should bring benefits as this central team can enforce, for example, security policies. But as we mentioned before, such a central approach often leads to a single team being the bottleneck for others. What happened to 'You build it, you run it'?

I love working in DevOps teams that are end-to-end responsible for their product. For me, that includes infrastructure provisioning as well. I've learned that there is no one-size fits all in our industry. Companies and teams are sometimes simply not ready for that way of working. Teams might not be cross-functional enough and have someone with enough knowledge to manage infrastructure. On other occasions, I see cloud implementations that are not mature enough to open up to development teams and ensure compliance and security. For those companies, this tool brings many benefits as it at least brings a lot of self-service options out of the box.
More mature teams can leverage the template definitions stored in source control and can easily be shared. That way, collaboration through inner sourcing can be promoted. Teams can start making small template changes and create a pull request. A platform team member can still be the code owner and must approve the change. Since a Dev Center can use multiple catalogs, mature teams can use their own GIT repository and link it. The use of the product and its way of working can, therefore, grow with the maturity of the engineers and the company. </>

Self-Guided Meditations With AI On Azure

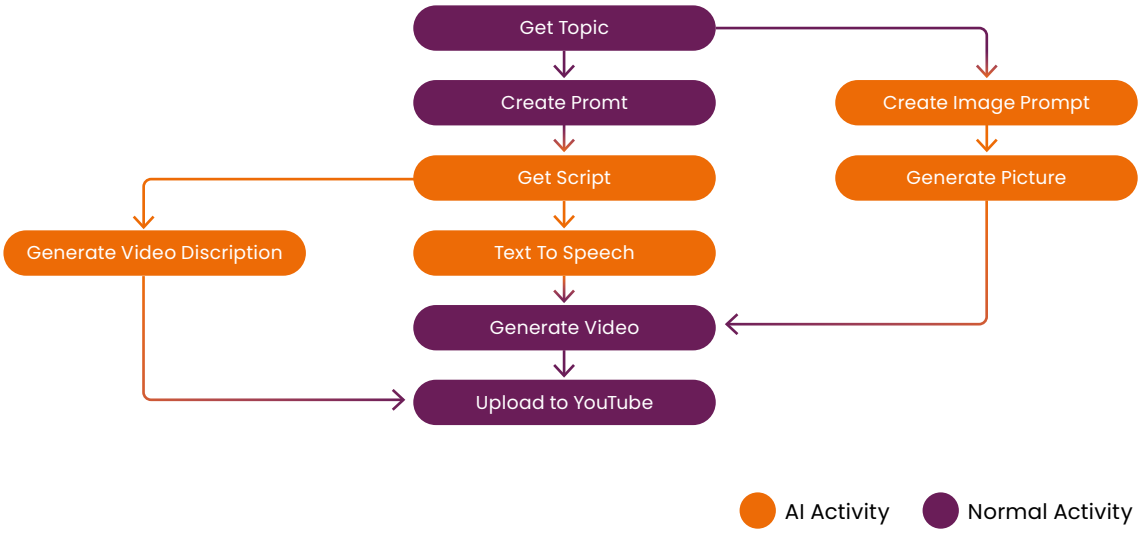
How to create unique content with Large Language Models

Author Matthijs van der Veer

Do you sometimes struggle with creating content? Do you fall victim to repetition? Whether it's a blog/manual/podcast you're trying to produce, Large Language Models can help you to create unique content if you use them correctly. In this post, I'll combine GPT-4, GPT-3.5, DALL-E, Azure Machine Learning and Azure AI Speech (formerly Cognitive Services) to create fresh daily content.

A self-guided meditation is usually an audio file where a narrator helps you focus on a topic. If you've ever tried meditation, you've probably used a mobile app like Calm or Headspace. These apps offer great content for self-guided meditations, but their offering is limited. This limitation is especially apparent if you like a specific style of meditation. I don't mind repetition, but if the same audio file is played twice, I become too aware of it to be effective. Surely, in the age of AI, I don't have to depend on humans to create my guided meditation. So, I set out to make an application to spin up a fresh session every day and upload it to YouTube. I had only two rules for myself:
1. No manual steps. I want everything to be automated.
2.A unique meditation is uploaded to YouTube every day.

Creating this type of content consists of multiple steps, perhaps more than you would imagine. Most of these steps use Artificial Intelligence and usually take some time to process. I'm building this in an Azure Durable Function to deal with these long-running processes. The code for this application can be found in my GitHub repository[1].



From topic to prompt

The first thing we need is a meditation instructor to create the script. Instead of relying on a human, we can ask a Large Language Model (LLM) to be our instructor. For our LLM, I've selected GPT-4. While we would get decent results with GPT-3.5 (and much faster, too!), GPT-4 seems more 'creative' in the content it generates, which is perfect if I need it to create long, unique scripts with little input. Later in this project, I'll use GPT-3.5 for more straightforward tasks. It's always important to take a moment to think about which model to use. Creativity is great, but GPT-4 costs roughly 20 times more than GPT-3.5!

To get our script, I created a prompt explaining what I expect from the LLM. This text is called a system prompt and will ground the rest of our conversation with the model. Here's an example of the prompt. The full version can be found in the repository:

[1] https://github.com/MatthijsvdVeer/PeaceProcessor

```
You are a friendly meditation instructor. You're going
to write a script for our next guided meditation.
You can advise your student to sit, with their hands
folded in their lap. They could sit on the ground, on
a chair or a pillow. Maybe they want to lie down.
Make sure to add breaks often, which can be between
5 and 40 seconds, depending on what feels natural.
Indicate a break in this format: <BREAK10> for 10
seconds, <BREAK40> for 40 seconds. The students love
it when you start them out focusing on their breath.
Help them breathe in through their nose and out through
their mouth. Repeat this exercise a few times. Add a
few-second break between the breaths.
After a few repetitions, we can focus on something else.
The user will supply you with the topic.
Please don't use the word "namaste". Don't add a break
at the end of the script. Address the student as
friend, student, but not plural. You can use the word
"you" to address the student. We're aiming for a
10 minute session, but don't mention that to the
student. Aim for around 1000 words.
```

My system prompt includes a few key things:

1. Giving the AI a role and some personality (i.e. a friendly meditation instructor).
2. It provides a few hints about the format of the meditation.
3. Instructions to add breaks.
4. Readies the AI to act on the user prompt.
5. Things to avoid.

The above is a very short prompt; through experience, I've learned that you spend about as much text to instruct the LLM what **not** to do as what you want it to do. And those cases (e.g. "don't mention the meditation time to the user") only come out once you're further in the process. Don't be afraid to experiment with your prompts. However, it's good to use professional tooling because prompt engineering is more than just stringing some words together. I used a new Azure Machine Learning feature called Prompt Flow to make my prompt.

Prompt Flow is a tool that allows you to iterate through prompts and measure their effectiveness easily. It's currently in preview but is worth a look. Not only does it allow you to create variants of prompts for manual testing, but it also allows for automated evaluation of the results of an LLM. You can score the results on how well they address the prompt, user input, or any other metric you can think of. I focused on how many words the LLM produces to evaluate my prompt. I would rather have a more extended meditation session, with more room to relax, than a fast one that doesn't get the point across.

Creating the script

With our prompt in hand, we can start asking GPT-4 to create our script. You can use GPT-4 through the OpenAI APIs or Azure OpenAI Service. The API specification is the same, but since the rest of my application runs on Azure, I prefer Azure OpenAI. It keeps all the billing in one place and offers extra features, like finetuning models, setting custom content filtering options, and more. I mentioned that our system prompt grounds the rest of the conversation. In this case, the conversation will be a short one. I supply a system prompt and then follow up with a user message. The user message will contain the topic of the meditation. When we submit this 'chat' to the API, the LLM will respond with our meditation script. It might seem strange that we need to submit this as a chat, but GPT-4 is not accessible through the Completion API (which will be deprecated in 2024). The code snippet below shows how a typical chat is built up in C#. Every message has a role: System, User, or Assistant. The System message contains our system prompt, the User message contains the topic and the LLM will respond with the Assistant message. Because we gave very few limitations in our system prompt, our topic can be a single word or an entire paragraph.

```
var prompt = await File.ReadAllTextAsync
("Prompts/script-prompt.txt");
Response<ChatCompletions> response = await
this.openAiClient.GetChatCompletionsAsync(
  "gpt-4",
  new ChatCompletionsOptions
   {
      Messages =
      {
          // Add the system prompt.
          new ChatMessage(ChatRole.System, prompt),

          // Add the user prompt containing the topic.
          new ChatMessage(ChatRole.User, createScript-
          Context.Topic)
      },
      Temperature = 0.8f,
      MaxTokens = 5000
  });
```

When you interact with the chat API, you can tweak several settings. The most critical settings for my scenario are the maximum amount of tokens and the "temperature". The maximum token amount tells the LLM how much text can be returned. Tokenisation could be an entirely separate article, so let's keep it simple: 1 token is not a fixed length and can be a single character, a syllable, or a word.

This can make it difficult if you need your content to be a specific length. The Temperature setting introduces more randomness to the response. Set it to 0 and the model will only respond with the most likely tokens. It will react with almost identical messages for the same input if you run it multiple times. Set the temperature to 1, and the responses will be wildly different to downright unpredictable.
Because I want to generate unique content without changing my system prompt every time, I set a relatively high temperature of 0.8. A high temperature allows the LLM to take any topic we provide, ranging from mindful to very silly, and turn it into a perfectly calming meditation.

Text to speech

We must transform this text into audio to make this a true self-guided meditation. There are a lot of text-to-speech applications out there, and I wanted one that sounded realistic enough to meditate on without being bothered by a computer voice. I settled on Azure AI Speech (recently renamed from Azure Cognitive Speech). Azure AI Speech hosts many speech-related features, including text-to-speech. The two main reasons I chose this service are that it's part of Azure, keeping all my resources/billing/access management in one place, and also because it supports Speech Synthesis Markup Language (SSML). This W3C standard for text-to-speech is beneficial for our type of content. You may have noticed in the prompt to GPT-4 we asked it to include breaks. SSML supports adding breaks in text-to-speech, which is excellent for meditations. You don't want someone droning in your ear continuously. SSML allows you to define the voice, choosing different languages and accents, as well as the style of the voice. Not all of the voices support different styles, I ended up with an American English voice that supported the "hopeful" style, which matches the style that I like during this type of content.

It's also worth noting that because SSML is a widely used standard, you can even ask GPT-4 to generate a script in SSML! This shows just how versatile GPT-4 is. While the results were good, in my testing with Prompt Flow I've noticed that it will reduce the overall duration of the meditation, even when you allow it to generate the maximum amount of tokens.
So instead, I had the LLM indicate breaks differently and turned it into SSML with some basic string formatting.

Now that I have some SSML, I can feed this to Azure AI Speech to get my audio!

```
var speechConfig = SpeechConfig.FromSubscription
(this.key, this.region);
speechConfig.SetSpeechSynthesisOutputFormat
(SpeechSynthesisOutputFormat.Riff16Khz16BitMonoPcm);

// Set the audio config to null. otherwise it will try
to use the default audio device.
// Pretty sure Azure data centers don't have a default
audio device.
using var speechSynthesizer = new SpeechSynthesizer
(speechConfig, null);
var result = await speechSynthesizer.SpeakSsmlAsync
(createNarrationContext.Script);
```

Add some background music

At this point, I thought I was done. I created a script and turned it into audio. But while listening to the audio, something was missing. The breaks go on for a long time (anything over 5 seconds is long), and I wondered if the session was over or if the software failed to produce a good result. But soon, I realised that the silence was just too distracting. To counter this, I wanted to add some background sounds. I found some royalty-free nature sounds online that were a perfect fit. After some quality time with the excellent 'NAudio' NuGet package and a deep dive into wave formats, I merged the narration with the background audio. This resulted in a perfect mix of a hopeful voice guiding you through the meditation and soothing nature sounds to fill the gaps.

Create Video

One of the goals I set for myself was to upload the results to YouTube. I wasn't quite sure if YouTube would allow an audio file to be uploaded, or at least a video file that was completely black, so I set out to add an image to the video track. Of course, I don't want to manually pick an image, nor do I want the same old image for every video I upload, so it's time for more artificial intelligence. Just in time, Microsoft released DALL-E as part of the Azure OpenAI service. DALL-E takes a text prompt and turns it into an image. Like with most AI tools, it's easy to get a good result, but difficult to get a great result. So, instead of creating a generic prompt, I asked GPT-4 to create the prompt for DALL-E instead. I supply the topic of the meditation, as well as this new system prompt just like I did to generate the script, and out rolls a prompt for DALL-E. I try not to limit the response too much, instead, I add some tips and tricks for using DALL-E.

```
Create a prompt for a square image. The prompt will
be sent to an AI algorithm that creates an image.
Only reply with the prompt.

The prompt should be for a colourful and calming image
for a meditation session. The user will supply you with
the topic. Here's some tips:
- Start with the character before the landscape, if
  there are characters involved, so you can get the
  body morphology right before filling the rest.
- The prompt should mention what's shown, as well as
  the colours and the mood.
- Adding an adjective like "gorgeous", "stunning" or
  "breathtaking" can make a big difference.
```

Figure 1: DALL-E prompt: A serene monk meditating under an ancient, sprawling tree, thoughts materializing as vibrant butterflies fluttering away, set against a backdrop of a breathtaking sunset, in the style of surrealism.

Publish to YouTube

I won't go into detail on how to upload videos to YouTube through a cloud application, but every video needs a video title and description. The original topic of a meditation session might not be suitable for a title, and it's usually not very descriptive, either. To automate the generation of this metadata, I decided to use GPT-3.5 Turbo. We don't need the creativity of the more advanced GPT-4, and with its low cost and increased generation speed, GPT-3.5 Turbo is a welcome addition to this application. I created two simple prompts to create a title and description. If I were to generate videos at scale, I would invest more time combining the title and description into one call to the API to save cost. The input for the LLM is my prompt, plus the topic of the meditation session; I also added the entire meditation script we got from GPT-4. All Large Language Models benefit from more context, and GPT-3.5 can use this script to create a comprehensive description. In this step, I also prompt the model to always end the description with the same line: "This content was created by Artificial Intelligence, and reviewed by humans." Let's talk about why that's necessary.

Time For Some Human Intervention

I set out on this mission to remove any manual actions, but I've applied artificial intelligence in almost every step of the process to create content for humans to enjoy. LLMs are exceptionally talented at producing some unexpected or even scary results. So it's time for a disclaimer: *You are responsible for its output*. Large Language Models are amazing, and will even seem like magic sometimes. Yet it doesn't "know" anything; it's just generating the most likely text, given your input as context.

For this reason, it is a human's job (that's you, dear reader) to verify the output of the models you use. In Azure Durable Functions, you can build this step in as an activity that won't be complete until you've sent the all-clear. But in my case, I upload the video to YouTube as private, and I listen to every single one before hitting that publish button. The benefit: I can now meditate every day. </>

In conclusion

Combining Large Language Models with text-to-speech and image generation can turn a single word into captivating audio or video content, I hope you will try to apply these applications of artificial intelligence yourself!

While building this application, I would call a URL in my function with the topic of the meditation. While great for testing, I didn't make this application with the idea of doing anything by hand. So, Who decides on these topics? Well, Artificial intelligence, of course.

Securing Azure Service Bus

Security should be considered from the initial stages of designing a product rather than as an afterthought. This is particularly important for Service Bus as it often forms a part of a larger system. Security requirements may vary depending on the use case; for instance, a banking solution would have different security needs compared to a solution for a local bakery.

Author Olena Borzenko

Let's examine common security risks, understand the importance of data encryption and various robust authentication methods such as Azure AD and shared access signatures, explore strategies for network protection, and emphasize the value of logging for enhanced oversight.

Data Protection and Risks

The sensitivity or potential impact of a data leak may be high when transmitting data via Service Bus, particularly if it involves financial transactions, medical records, or sensitive personal information. It is crucial to protect the data from risks such as data exfiltration, unauthorized data movements, and unauthorized access. It is also important to have proper logging to monitor what is happening with the data.

Service Bus performs *encryption in transit*, or in other words, it ensures that data is encrypted while being transmitted. This includes encryption when data is moving from the client to Service Bus, within Service Bus, and from Service Bus to the consumer.
By default, Azure Service Bus supports TLS 1.2 protocol on public endpoints. Initially, it was TLS 1.0, but due to customer demands for higher security, it now defaults to the higher version. However, this doesn't mean that versions 1.0 and 1.1 are deprecated; they are still supported for backward compatibility, and users can set a minimum TLS version in their name-space. During other exchanges, secure protocols like HTTPS for straightforward RESTful operations and AMQP for efficient message queuing are used.

Besides encryption in transit, Service Bus also performs **encryption at rest**, meaning messages are encrypted while they are at rest (stored).
This process is done automatically, and users don't have to do anything to enable it. The encryption uses Azure storage encryption, and Service Bus is transitioning to service fabric storage for improved performance and cost savings.

But what if the built-in security layers are not sufficient to meet customer requirements? In such cases, users can enhance security by bringing their own encryption key, stored in Azure Key Vault — a method commonly referred to as **BYOK (Bring Your Own Key)**.
The provided key can be used to encrypt data, adding an extra layer of security. This is particularly important for organizations with stringent security policies.

So far we've examined some built-in security features as well as the method of introducing an extra layer of protection using the BYOK approach. There are also actions that can be taken on the client side for more advanced scenarios.

For example, an additional layer of encryption can be implemented by the client, an approach we can refer to as **client-side encryption**. The data protection step is performed before sending the data to Service Bus.
While this is the most secure method, it also requires the most effort, as the client is responsible for both encryption and decryption.
This approach is commonly used in highly sensitive environments like healthcare, where data breaches can have significant consequences.

As we can see, there are many different mechanisms to secure our data. For maximum security, we can go a step further and opt for **multi-layer encryption**.
By combining client-side encryption, bringing your own key, and the platform encryption provided by Service Bus, we can achieve the highest level of data protection.

Authentication methods

As previously mentioned, Azure Service Bus offers two types of authentication: **Azure Active Directory (Azure AD)** and **Shared Access Signatures (SAS) keys**. Let's take a brief overview of these two types to understand what might be more suitable for certain needs.

As a more modern and recommended form of authentication, Azure Active Directory (Azure AD) offers a range of features that enhance security and ease of management. It supports various types of accounts, service principals and provides a streamlined and secure method for managing identities. Its flexibility makes it easier to manage access for different clients or customers. For those looking to further tighten security, it's possible to disable SAS authentication entirely and rely solely on Azure AD.
Additionally, custom roles can be created to offer more granular permissions, allowing for tailored access control based on specific needs.

Another robust authentication option, known as Shared Access Signatures (SAS) keys, involves generating a connection string from primary and secondary keys for authentication. These keys can be set at different scopes — namespace, topic, or queue — to allow fine-grained access control.
To serve various consumers or users you can also create multiple keys but it's important to note that **they are static and require manual rotation** for enhanced security, especially the root manager key that controls the entire namespace. For extra security, using a token provider, such as an API that issues authentication tokens, is recommended over direct key usage.
Although SAS are somewhat dated, they remain supported and are useful for systems restricted to this authentication method.

Network-level security
Having explored data protection measures and authentication methods, let's now turn our attention to another crucial aspect of securing Azure Service Bus: **network-level security**.

One effective measure is to set **Service Tags** on the Service Bus namespace, which allow you to specify which Azure services can access your Service Bus. Additionally, IP Filtering can be employed to limit access to specific IP addresses or ranges. For those using the premium tier of Service Bus, adding the Service Bus to a Virtual Network can further minimize the attack surface.

It's worth mentioning that Service Bus is a foundational element of Azure's architecture and offers tier-specific features. For instance, the premium tier provides advanced options like VNet integration, mainly because it operates on a dedicated resource model, unlike the standard tier.

Who knows, maybe in the future the gap between the two tiers will be bridged to some extent. But for now, this gap results in a significant price difference between the standard and premium plans. Despite the use of dedicated hardware resources like virtual machines in the premium service, efforts are underway to narrow this price difference and make the subscription more accessible and affordable. Additionally, guidance and templates may be introduced to help determine the continuous need for the service or its occasional use.

Logging and monitoring for security and system health
As we conclude our comprehensive exploration of Azure Service Bus security, let's delve into the indispensable aspects of logging and monitoring for both security and overall system health.

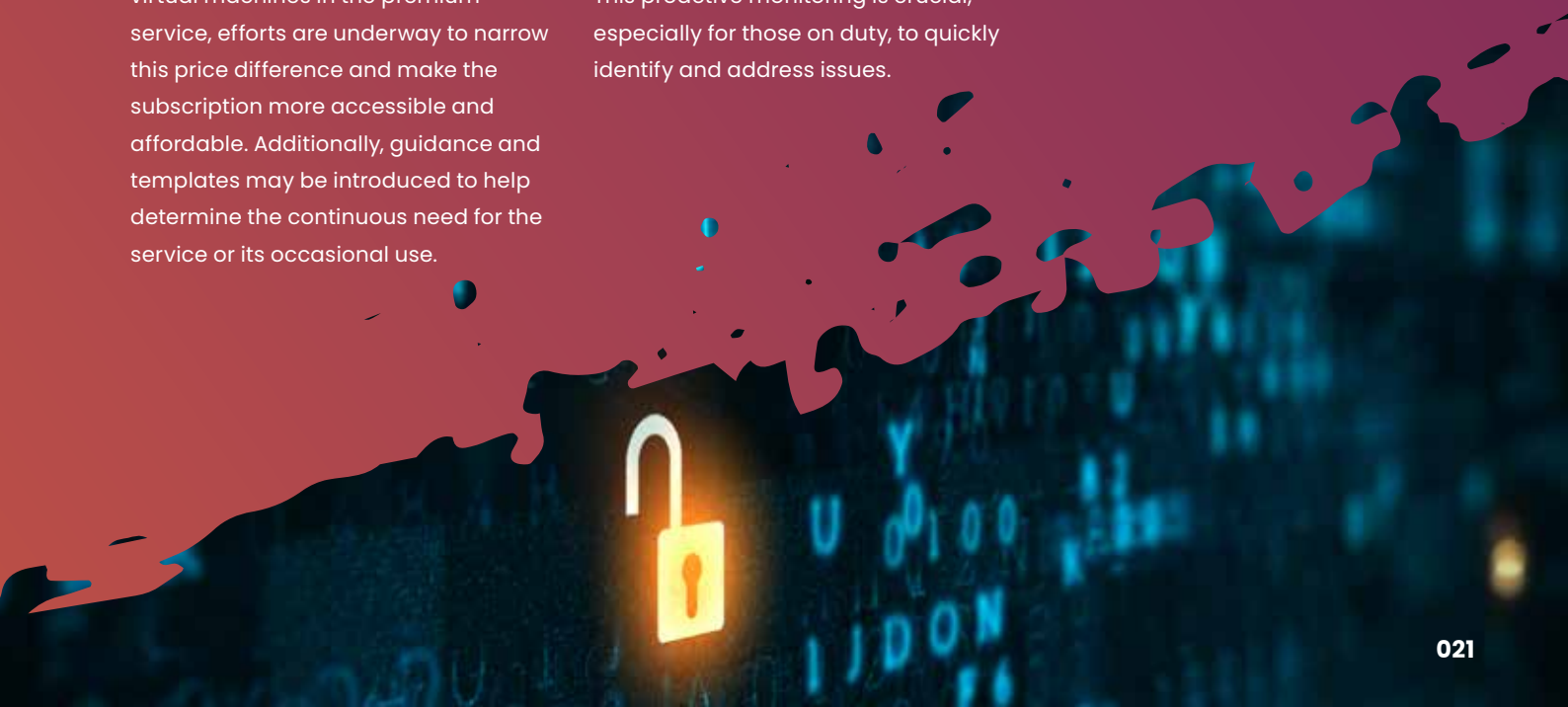
Service Bus generates a significant amount of logs, accessible through Application Insights and Log Analytics. The Kusto Query Language (KQL) is particularly useful for those who wish to work with these logs, as they include information about messages sent, connections made, and operations performed.

There is also support for Azure Policy, which allows users to set policies for various configurations. For example, a user can set a minimum TLS version across all subscriptions to ensure that security standards are met. This helps ensure that everyone adheres to the same security principles.

It is important to not only log information but also to actively monitor it for anomalies or issues. Service Bus allows users to set up alerts based on certain conditions or dynamic thresholds. For instance, if there is an unusual spike in connections, an alert can be triggered. This proactive monitoring is crucial, especially for those on duty, to quickly identify and address issues.

Through Azure Monitor, users can integrate with other services such as Logic Apps or Azure Functions. Some companies have automated their workflow such that when an alert is triggered, the system analyzes what's happening, assigns it to the correct team, sets a priority, and creates a ticket. This streamlines the process and ensures that the right people can start working on the issue promptly.

In summary, the level of security implementation should be tailored to the specific scenario, taking into account the criticality of the data and operations involved. For instance, a small customer sending a few messages may not need the same robust measures as a large organization handling sensitive data. Alongside this, configuring encryption is a pivotal step, with options like client-side encryption providing added assurance by keeping keys on-premises. While Azure is compliant with GDPR and other standards, it's essential to verify these, especially when dealing with sensitive information. </>



Azure Policy Unveiled: Ignite Your Cloud Management Passion

Imagine your company having a multitude of Azure resources, and you want to ensure all of them are compliant with your company's standards. You could go through each resource and check if they are compliant, but that would be a lot of work. Luckily, Azure Policy can help you with that. Azure Policy is a management tool that helps you enforce and control the settings and configurations of resources within your Azure cloud environment. It enables you to define and enforce rules and policies to ensure that your resources adhere to specific compliance and governance requirements. These policies can cover various aspects, such as security, resource tagging, and naming conventions, helping you maintain a consistent and secure cloud infrastructure. Azure Policy provides a centralized way to monitor and enforce these policies, ensuring that your Azure resources are aligned with your organization's standards and best practices.

Author **Patrick de Kruijf**

Azure Policy works with definitions to set the conditions and rules to be executed. Definitions dictate the logic, followed by assignments that apply the logic to a scope. A scope can be a management group, subscription, resource group, or resource. When an assignment is made, you can review the compliance of your resources in the compliance dashboard.

When you are ready to look at Azure Policy, you will probably be overwhelmed. Luckily, Microsoft Azure has supplied you with a set of built-in policies you can use to get started. Additionally, when the built-in policies are not sufficient for your needs, you have the option to create custom policies.

Below a short summary of the differences:

- **Built-in** policy definitions are provided by Microsoft and can be used to audit against your environment.
- **Custom** policy definitions are created by you and can be used to audit against your environment.

Important note for built-in policies, they are deployed to the Root Tenant Group and their names are GUIDs. Their display name will explain better what each policy definition does.

Figure 1: Compliancedashboard

Why would I use it?

Azure Policy is quite powerful and enables your organization to enforce standards and assess compliance. It also helps to bring your resources to compliance through bulk remediation for existing resources and automatic remediation for new resources. But why would you use it? Let's take a look at some of the benefits of using Azure Policy:

- **Enforce standards**: Azure Policy helps to enforce standards and assess compliance. You can also use policies to prevent or (automatically) remediate non-compliant resources.
- **Centralized management**: Azure Policy provides a centralized management experience for all your policies. You can create, assign, and manage policies from a single location.
- **Aggregated view**: Azure Policy provides an aggregated view of the state of your environment through the compliance dashboard, which shows the overall state of the environment and allows you to view the state of individual resources or policies.
- **Built-in compliance standards**: Azure Policy provides built-in compliance standards that can be used to audit against your environment. These standards include CIS, PCI, HIPAA, ISO, NIST, SOC, and more.
- **Bring resources to compliance**: Azure Policy helps to bring your resources to compliance through bulk remediation for existing resources and automatic remediation for new resources.
- **Custom compliance standards**: Azure Policy provides the ability to create custom compliance standards that can be used to audit against your environment.
- **Audit and remediate**: Azure Policy provides the ability to audit and remediate your environment. You can audit your environment by using the 'audit' effect. You can remediate your environment by using the 'deployIfNotExists' effect.
- **Steer user behavior**: Azure Policy can be used to steer user behavior by restricting the use of specific resource types. For example, you can restrict the use of public IP's.

Now we know why we would use Azure Policy, let's take a look at some of the real life scenario's that can be solved with Azure Policy:

- **Restricting the location of resources** all resources should be deployed in West Europe or North Europe
- **Enforcing tagging on resources** all resources should have a cost center tag
- **Steer user behavior by restricting the use of specific resource types or SKUs** no use of public IP's, deny creation of GPU VM's
- **Enforcing the configuration of specific resource configurations** soft delete on key vaults, or encryption on storage accounts
- **Configure DNS Private Zone settings automatically on private endpoints** configure the DNS Private Zone settings on private endpoints to use the private DNS zone
- **Enforce resource naming for Resource Groups** all resource groups should start with rg-

How do I use Azure Policy?

First, you start with a policy definition that dictates the logic to execute. This can be either a built-in or custom policy. Built-in policies are deployed to the Root Tenant Group and therefor available everywhere in your Management Group and Subscription hierarchy. Custom policies are deployed to a specific scope and therefor only available in that scope or a child scope. The deployment of a policy definition is nothing more than making the policy definition available to be used.

In order to apply the logic of a policy definition to a certain scope, which can be the scope the policy definition was deployed to or any of the child scope in the hierarchy. You will need to create a policy assignment. A policy assignment is the actual assignment of the policy definition to a scope. The policy assignment will evaluate the resources in the scope against the policy definition. The policy assignment will also show the compliance of the resources in the scope.

But what if I want to combine multiple policy definitions and assign it as part of a single policy? This is where policy initiatives come in. Policy initiatives are a collection of one or more policy definitions. Like policy definitions, these are deployed to a specific scope and can be assigned to a scope. The assignment of a policy initiative is the same as the assignment of a policy definition.

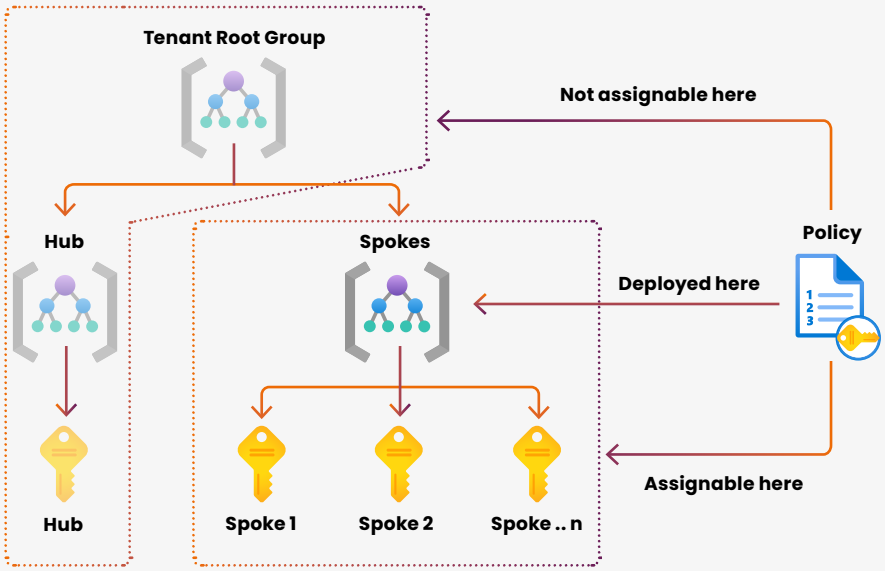


Figure 2: Policy deployment scope and assignment

Policy Definitions

A policy definition is a JSON document that defines the logic of the policy. Let's create a policy definition together.

- **Name**: The name of the policy definition.
- **Description**: A description of the policy definition.
- **PolicyRule**: The policy rule that defines the logic of the policy definition.
- **Metadata**: Metadata about the policy definition.
- **Parameters**: Parameters that can be used in the policy rule.

Name

The name of the policy definition. The name is used to identify the policy definition. The name must be unique within the scope of the policy definition.

Description

A description of the policy definition. The description is used to describe the policy definition. The description is optional.

PolicyRule

Policy definitions use policy rules to dictate the logic to perform or validate. These rules are built using an if-then construct. The **IF** part contains the resource(s) to search for, the **THEN** part contains action to take.

The IF-part of the policy rules

The **IF** parts contain multiple options to identify the scope for the policy. These use logical operators to check the conditions. Within the logical operators, conditions are used to determine if the policy should execute the **THEN** part. You can use the following operators:

- **not** - the conditions should not be true (inverting the result).
- **allOf** - each condition in the block should be true.
- **anyOf** - any condition in the block should be true.

With these logical operators you are flexible in terms of identifying which resources should be used in the specific policy. You can even use nested operators. The logical operators use conditions to identify when to perform the **THEN** part of the policy. Conditions are always described with a field value and an option that returns a true or false result. Possible options are:

- **equals** – true if the field value matches the equals value.
- **notEquals** – true if the field value does not match the notEquals value.
- **exists** – true if the field value exists.
- **in** - true if the field value is in the list of values.

The field values can contain a multitude of different options. Let's go over some of the commonly used options:

- **Type** indicates the actual resource type (i.e., "Microsoft.KeyVault/vaults")
- **Location** indicates the resource location (i.e., "WestEurope")
- **Id** indicates the actual resource ID of an Azure resource (see properties on a resource to view/copy the resource ID)
- **(https://learn.microsoft.com/en-us/azure/governance/policy/concepts/definition-structure)** can be used to access a property of a resource type.
- And many more **(https://learn.microsoft.com/en-us/azure/governance/policy/concepts/definition-structure)**.

The THEN-part of policy rules

For the THEN part, Azure Policy uses an effect to identify the action to be taken when the conditions of the definition or initiative are non-compliant. Eight types of effects are available:

- **Deny** will ensure the non-compliant resource cannot be created or deployed.
- **Audit** will audit the resources' compliance and show the status in the compliance dashboard.

- **DeployIfNotExists** will deploy the configuration specified in the definition to the resource if the configuration does not exist.
- **AuditIfNotExists** will audit the configuration specified in the definition and only report if the resource configuration does not exist.
- **Modify** is used to add, update, or remove properties or tags on a subscription or resource during creation or update.
- **Append** is used to add more fields to the requested resource during creation or update.
- **Manual** enables you to self-attest the compliance of resources or scopes.
- **Disabled** means the logic in the definition will effectively do nothing and is turned off.

Metadata

This is metadata about the policy definition. The metadata is used to provide additional information about the policy definition. The metadata is optional. Common metadata properties are:

- **Category**: The category of the policy definition.
- **Version**: The version of the policy definition.
- **Deprecated**: Indicates if the policy definition is deprecated.
- **Preview**: Indicates if the policy definition is in preview.

Parameters

Parameters are optional. They can be used to make the policy definition more flexible. Parameters are defined in the parameters section of the policy definition. Parameters are referenced in the policy rule by using the following syntax: (`parameterName`). Parameters will be prompted for when assigning the policy definition.

Result

Let's take a look at an example of a policy definition.
The following policy definition is used to audit the use of the 'Microsoft.Storage/storageAccounts/networkAcls/defaultAction' property. The policy definition is named 'audit-storage-account-network-acl-default-action' and has the following properties:

- **Description**: Audit the use of the `Microsoft.Storage/storageAccounts/networkAcls/defaultAction` property.
- **PolicyRule**: If the `type` of the resource is `Microsoft.Storage/storageAccounts` and the `Microsoft.Storage/storageAccounts/networkAcls/defaultAction` property is not equal to `Deny`, then audit the resource.
- **Metadata**: The category is `Storage` and the version is `1.0.0`.

```
{
  "properties": {
    "displayName": "Audit storage account network ACL
    default action",
    "description": "Audit the use of the Microsoft.
    Storage/storageAccounts/networkAcls/defaultAction
    property.",
    "mode": "Indexed",
    "policyRule": {
      "if": {
        "allOf": [
          {
            "field": "type",
            "equals": "Microsoft.Storage/storageAccounts"
          },
          {
            "field": "Microsoft.Storage/storageAccounts/
            networkAcls/defaultAction",
            "notEquals": "Deny"
          }
        ]
      },
      "then": {
        "effect": "audit"
      }
    },
    "metadata": {
      "category": "Storage",
      "version": "1.0.0"
    }
  }
}
```

Examples

I only want to allow resources to be deployed in the West Europe region.
This can be achieved by using the **DENY** effect, it will verify the condition and it will **DENY** the deployment if the condition returns as true.

In this case **IF** the location is not equal to WestEurope returns true **THEN** we **DENY** the deployment.

```
"policyRule": {
  "if": {
    "not": {
      "field": "location",
      "equals": "WestEurope"
    }
  },
  "then": {
    "effect": "Deny"
  }
}
```

I want to always add soft delete on key vaults on creation
For this scenario we can use **APPEND** as the effect, we will verify if the condition is true and then we will append a specific configuration/property.

In this case, we verify **IF** the type of the resource is a Key Vault and if the soft delete option is not true. When the conditions have been verified, we will **THEN APPEND** the soft delete option.

```
"policyRule": {
  "if": {
    "anyOf": [
      {
        "allOf": [
          {
            "field": "type",
            "equals": "Microsoft.KeyVault/vaults"
          },
          {
            "field": "Microsoft.KeyVault/vaults/
            enableSoftDelete",
            "notEquals": true
          }
        ]
      }
    ]
  },
  "then": {
    "effect": "append",
    "details": [
      {
        "field": "Microsoft.KeyVault/vaults/
        enableSoftDelete",
        "value": true
      }
    ]
  }
}
```

How do I deploy policy definitions?
As always with cloud resources, the preferable way to deploy them is using Infrastructure as Code (IaC). Azure Policy definitions can be deployed using ARM templates, Azure CLI, Bicep, Terraform, or PowerShell. The following example shows how to deploy the policy definition using a PowerShell script:

```
New-AzPolicyDefinition -Name 'audit-storage-account-
network-acl-default-action' -DisplayName 'Audit storage
account network ACL default action' -Description
'Audit the use of the Microsoft.Storage/storageAccounts/
networkAcls/defaultAction property.' -Policy 'audit-
storage-account-network-acl-default-action.json'
-Mode All
```

Important to note is the PowerShell cmdlet uses the `-Policy` argument to define the Policy Rule. Each of the important parts of the policy definition is defined in the cmdlet.

File: audit-storage-account-network-acl-default-action.json:

```
{
  "if": {
    "anyOf": [
      {
        "allOf": [
          {
            "field": "type",
            "equals": "Microsoft.KeyVault/vaults"
          },
          {
            "field": "Microsoft.KeyVault/vaults/
            enableSoftDelete",
            "notEquals": true
          }
        ]
      }
    ]
  },
  "then": {
    "effect": "append",
    "details": [
      {
        "field": "Microsoft.KeyVault/vaults/
        enableSoftDelete",
        "value": true
      }
    ]
  }
}
```

Policy Initiatives

A policy initiative is a collection of one or more policy definitions. Policy initiatives are used to group policy definitions together. This can be useful when you want to assign multiple policy definitions to a scope. Instead of assigning each policy definition individually, you can assign the policy initiative. The policy initiative will then assign all the policy definitions that are part of the policy initiative. Let's create a policy initiative together.

```
{
  "properties": {
    "displayName": "Audit storage account network ACL",
    "description": "Audit the use of the Microsoft.
    Storage/storageAccounts/networkAcls/defaultAction
    property and the Microsoft.Storage/storageAccounts/
    networkAcls/bypass property.",
    "metadata": {
      "category": "Storage",
      "version": "1.0.0"
```

```
    },
    "policyDefinitions": [
      {
        "policyDefinitionId": "/providers/Microsoft.
        Authorization/policyDefinitions/audit-
        storage-account-network-acl-default-action",
        "parameters": {}
      },
      {
        "policyDefinitionId": "/providers/Microsoft.
        Authorization/policyDefinitions/audit-
        storage-account-network-acl-bypass",
        "parameters": {}
      }
    ]
  }
}
```

How do I deploy policy initiatives?

As always with cloud resources, the preferably way to deploy them is using Infrastructure as Code (IaC). Azure Policy initiatives can be deployed using ARM templates, Bicep, Terraform, or PowerShell. The following example shows how to deploy the policy initiative using Powershell:

```
# Create a new policy initiative using the policy
definitions supplied in the audit-storage-account-
network-acl.json file
New-AzPolicySetDefinition -Name 'audit-storage-
account-network-acl' -PolicyDefinition 'audit-
storage-account-network-acl.json'
```

File: audit-storage-account-network-acl.json:

```
[
  {
    "policyDefinitionId": "/providers/Microsoft.
    Authorization/policyDefinitions/audit-storage-
    account-network-acl-default-action",
    "parameters": {}
  },
  {
    "policyDefinitionId": "/providers/Microsoft.
    Authorization/policyDefinitions/audit-storage-
    account-network-acl-bypass",
    "parameters": {}
  }
]
```

Policy Assignments

A policy assignment is the actual assignment of the policy definition or policy initiative to a scope. The policy assignment will evaluate the resources in the scope against the policy definition or policy initiative. The policy assignment will also show the compliance of the resources in the scope. Let's create a policy assignment together, using PowerShell.

```
# Get the subscription data
$Subscription = Get-AzSubscription -SubscriptionName
'Subscription01'
# Get the policy definition data
$Policy = Get-AzPolicyDefinition -Name 'audit-storage-
account-network-acl-default-action'
# Create the policy assignment using the retrieved
subscription and policy definition data
New-AzPolicyAssignment -Name 'audit-storage-account-
network-assignment' -PolicyDefinition $Policy -Scope
"/subscriptions/$($Subscription.Id)"
```

How can I see my resource compliance?

After you've have created and assigned your policies, you can view the compliance of your resources. The compliance dashboard provides an aggregated view of the state of your environment. It shows the overall state of the environment and allows you to view the state of individual resources or policies. The compliance dashboard can be found in the Azure Portal under **All services** > **Policy** > **Compliance**.

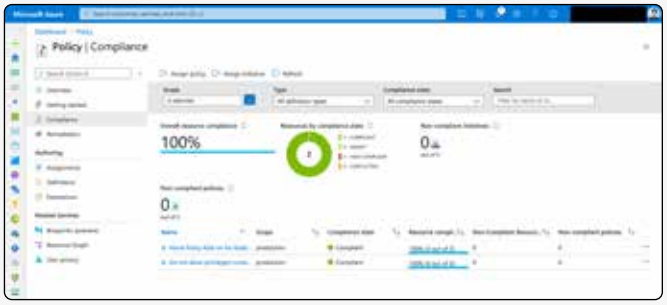


Figure 3: Compliancedashboard

Regulatory compliance

In order to view regulatory compliance, Microsoft Azure also uses Azure Policy to report on the compliance state of the regulatory compliance standards you have assigned. Whenever you select a regulatory compliance standard, Azure Policy will automatically create a policy assignment to audit the compliance state of the regulatory compliance standard. Azure Defender for Cloud also uses the input from Azure Policy to show recommendations in the Azure Portal.

To view the compliance state of the regulatory compliance standards you have assigned, you can use the regulatory compliance dashboard. The regulatory compliance dashboard can be found in the Azure Portal under **All services** > **Microsoft Defender for Cloud** > **Regulatory compliance**.

Figure 4: Regulatory Compliance Dashboard

How does remediation work?

Remediation is the process of bringing a non-compliant resource into compliance. When remediation is done manually, you can trigger the remediation from the compliance dashboard. When remediation should be done automatically, which is only possible when using the **DeployIfNotExists** or **Modify** effect, you can configure the policy to automatically remediate non-compliant resources. In order for the policy to remediate automatically, it will use a managed identity. This managed identity should be supplied in the policy assignment. In order to use the remediation, you need to specify, in the policy definition, which role the managed identity should have on the resource. The policy assignment will start the deployment to perform the remediation task. When using the **DeployIf-NotExists** effect, the deployment will be visible in the deployment dashboard so you can track and troubleshoot the deployment.

Automatic remediation

Let's use a built-in policy definition to show how automatic remediation works. The built-in policy definition is named 'Add or replace a tag on resource groups' and has the following PolicyRule logic:

```
{
  "if": {
    "allOf": [
      {
        "field": "type",
        "equals": "Microsoft.Resources/subscriptions/
        resourceGroups"
      },
      {
        "field": "[concat('tags[', parameters('tagName'),
        ']')]",
        "notEquals": "[parameters('tagValue')]"
      }
    ]
  },
  "then": {
    "effect": "modify",
    "details": {
      "roleDefinitionIds": [
        "/providers/microsoft.authorization/role-
        Definitions/b24988ac-6180-42a0-ab88-20f7382dd24c"
      ],
      "operations": [
        {
          "operation": "addOrReplace",
          "field": "[concat('tags[', parameters
          ('tagName'), ']')]",
          "value": "[parameters('tagValue')]"
        }
      ]
    }
  }
}
```

The policy definition will check if the resource type is a resource group and if the tag is not equal to the specified value. If the conditions are true, the policy will modify the resource group and add or replace the specified tag.

As described earlier, we should create a policy assignment to make this logic active on a certain scope. Let's create a policy assignment together, using PowerShell.

```
# Get the subscription data
$Subscription = Get-AzSubscription -SubscriptionName
'Subscription01'
# Get the policy definition data, using a Where-
Object, since built-in policies are named with a GUID
$Policy = Get-AzPolicyDefinition | Where-Object
{$_.Properties.DisplayName -eq 'Add or replace a
tag on resource groups'}
# Define the parameters for the policy assignment to
pass to the -PolicyParameterObject parameter
$parameters = @{'tagName'='Environment';'tagValue'
='Production'}
# Create the policy assignment using the retrieved
subscription and policy definition data
New-AzPolicyAssignment -Name 'add-tag-resource-group'
-PolicyDefinition $Policy -Scope
"/subscriptions/$($Subscription.Id)" -IdentityType
'SystemAssigned' -Location 'WestEurope'
-PolicyParameterObject $parameters
```

Figure 5: Policy Assignment - Remediation

The policy assignment will automatically remediate the resource group and add or replace the specified tag. The deployment will be visible in the deployment dashboard so you can track and troubleshoot the deployment.

So what happens when you create a resource group without the specified tag? Let's try it out!

```
# Set the subscription context
Set-AzContext -Subscription 'Subscription01'
# Create a new resource group without the specified tag
New-AzResourceGroup -Name 'rg-remediation-test'
-Location 'WestEurope'
```

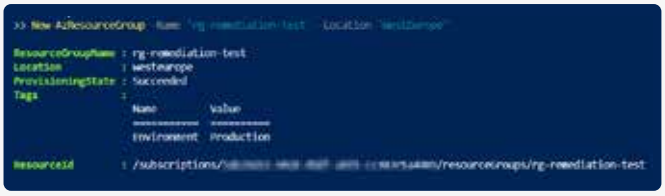


Figure 6: Result of resource group

As you can see, the resource group is created without the specified tag and the output already shows the policy assignment has remediated the resource group. When we check the resource group, we can see the tag is added.

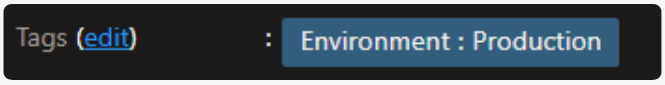


Figure 7: Tag present

Manual remediation

Let's use the same example as with the automatic remediation, but we have now changed the policy to seek the 'TagName:Demo' with a 'TagValue:ManualRemediation'.

```
# Get the subscription data
$Subscription = Get-AzSubscription -Subscription-Name
'Subscription01'
# Get the policy definition data, using a Where-
Object, since built-in policies are named with a GUID
$Policy = Get-AzPolicyDefinition | Where-Object
{$_.Properties.DisplayName -eq 'Add or replace a
tag on resource groups'}
# Define the parameters for the policy assignment to
pass to the -PolicyParameterObject parameter
$parameters = @{'tagName'='Demo';'tagValue'=
'ManualRemediation'}
# Create the policy assignment using the retrieved
subscription and policy definition data
New-AzPolicyAssignment -Name 'add-tag-resource-group'
-PolicyDefinition $Policy -Scope "/subscriptions/
$($Subscription.Id)" -IdentityType 'SystemAssigned'
-Location 'WestEurope' -PolicyParameterObject
$parameters
```

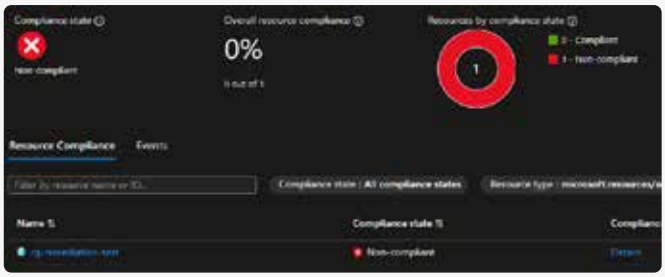


Figure 8: Policy Assignment - Non Compliant

Since the policy is not compliant, we can manually remediate the resource group by clicking on the **Create Remediation Task** button in the assignment page on the subscription.

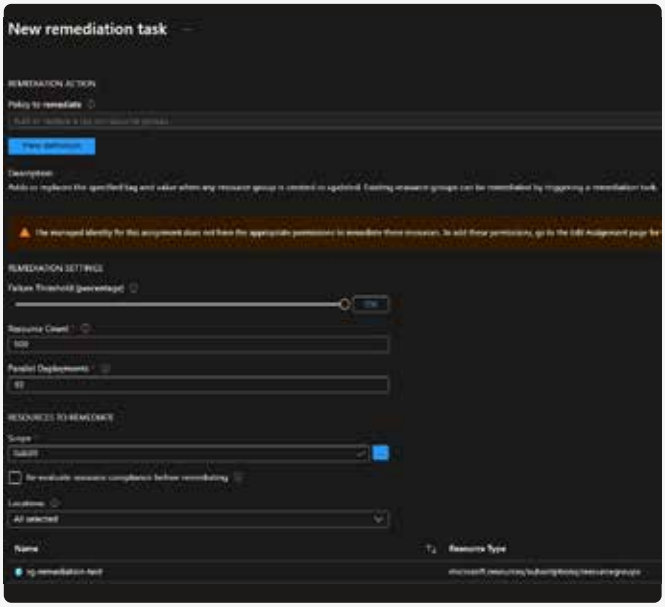


Figure 9: Manual remediation input

After the remediation task has been created, we can see the task in the remediation tasks overview. And you should even see a completed remediation task.

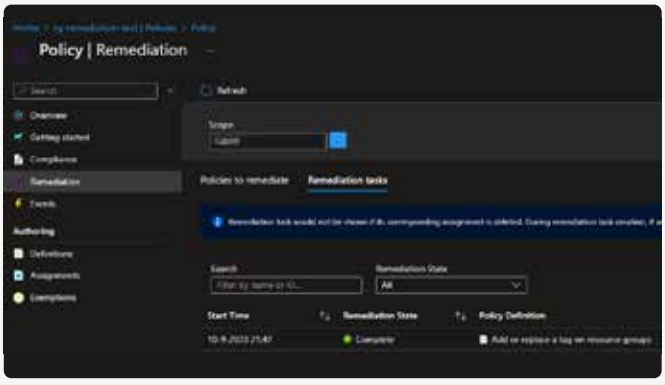


Figure 10: Completed remediation

Exemptions

Sometimes you want to exclude certain resources from being evaluated by a policy. This can be done by using exemptions. Exemptions can be made on policy assignments and on individual resources. Exemptions can be made for a specific amount of time or indefinitely. Exemptions can be made for the following reasons:

- **Mitigation**: The resource is already mitigated.
- **False positive**: The resource is evaluated as non-compliant but it is compliant.
- **Business justification**: The resource is evaluated as non-compliant but it is compliant for business reasons.
- **Waiver**: The resource is evaluated as non-compliant but it is compliant for legal reasons.

How do I create an exemption?

Exemptions can be created using the Azure Portal, PowerShell, Azure CLI, or REST API. Let's create an exemption using PowerShell.

```
# Get the subscription data
$Subscription = Get-AzSubscription -SubscriptionName
'Subscription01'
# Get the policy assignment data
$PolicyAssignment = Get-AzPolicyAssignment -Name
'add-tag-resource-group' -Scope "/subscriptions/
$($Subscription.Id)"
# Create the exemption
New-AzPolicyExemption -Name 'exemption-add-tag-
resource-group' -PolicyAssignment $PolicyAssignment
-ExemptionCategory 'Waiver' -ExpiresOn (Get-Date)
.AddDays(7)
```

[1] https://www.azadvertizer.net/

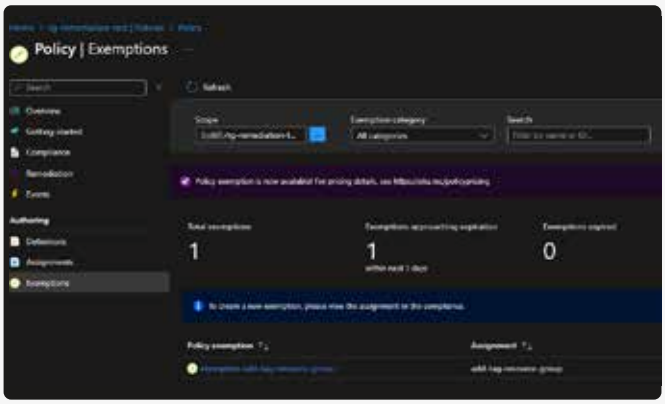


Figure 11: Exemption created

Now what?

Now that you know what Azure Policy is and how to use it, you can start using it in your own environment. Start with the built-in policies and see if they fit your needs. If they don't, you can always create custom policies.

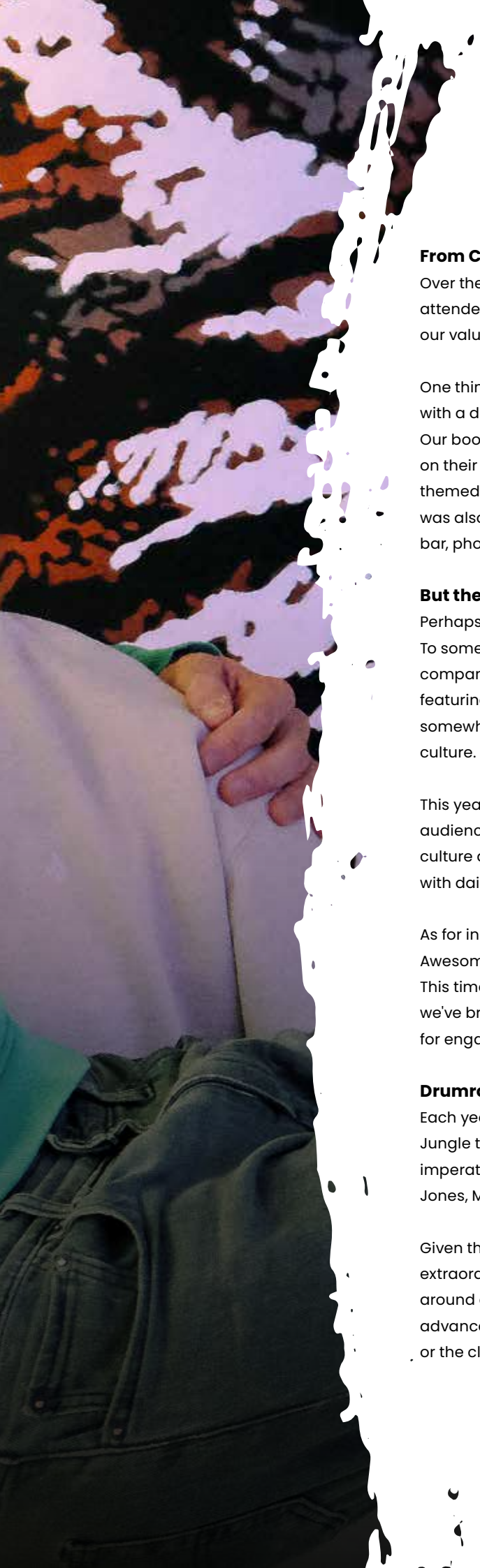
If you want to explore and view anything policy related in the Azure Portal, simply go to **All services** > **Policy**. Here you can view the compliance dashboard, policy definitions, policy initiatives, policy assignments, and exemptions.

If you want to know which Azure Policies are available built-in, AzAdvertizer[1] is a great resource to view all the built-in policies. </>

The Making Of: The Xebia | Xpirit Techorama Escape Room

At Xebia | Xpirit, we're a passionate team always eager to learn, grow, and innovate. When it comes to our outings at various events, there's one that stands out: Techorama. We attend both its Belgium edition in May and the Netherlands edition in October.

Authors Thijs Limmen & Dennis Thie (with guest appearances by Natascha Former and Olaf Walther)



From Caffeine to Culture: Xebia | Xpirit at Techorama

Over the years, we've tried to bring something new and refreshing to Techorama attendees. Our booths have transitioned from simple setups to ones that reflect our values, culture, and love for innovation.

One thing though, has been a common theme. It is well-acknowledged that with a deep-technical conference comes a demand for high levels of energy. Our booths have always offered premium coffee, ensuring attendees can load up on their caffeine levels. Part of our booth last year, in 2022, was a Las Vegas themed barista corner. Not only did we serve great coffee, but our booth concept was also in the style of an American coffee bar with a cozy seating area, barista bar, photo wall and even a giant rotating logo. All in all, it was a great experience!

But then, how can we do better than last year?

Perhaps the American-style coffee bar concept was too convincingly executed. To some, it wasn't immediately clear that we're not just a cozy café, but a company, one that's hiring. Additionally, despite having such a standout booth, featuring multiple speakers, and a significant presence as attendees, we were somewhat muted. We didn't actively scout the talent that resonated with our culture. In a way, we seemed content being the industry's best-kept secret.

This year, our objective is sharper. We aim to align more closely with our target audience by showcasing what truly defines us. Everything from our company culture and core values to our proudest achievements and the tools we interact with daily, like DevOps, Azure, and GitHub will be in the spotlight.

As for innovation, Olaf, our creative mastermind, and designated Chief of Awesomeness, has consistently delivered remarkable ideas year after year. This time, we've collaborated even more closely with him. Right from the start, we've brainstormed an interactive game concept, allowing us a broader canvas for engagement.

Drumroll....This year's Techorama theme: Jungle

Each year, Techorama introduces a unique theme. With the adoption of the Jungle theme for this year's edition, we had plenty food for thought! But it was imperative to think beyond conventional interpretations, such as Tarzan, Indiana Jones, Mayan temples, bananas, and the Jungle Book.

Given the theme, we set out to ideate with a few colleagues. What could be an extraordinary approach to this theme? Our vision quickly became centered around an abandoned airplane in the jungle - a symbol of technological advancements halted in its trajectory. And a small wink towards to the skies -- or the cloud.

Like any project... the first design iterations

Our creative mastermind and Chief of Awesomeness, Olaf, needed nothing more. He sketched out the plane fuselage concept, catering for our barista section and space for the Escape Room experience. Knowing very well we'd have to take it apart for transport and then put it back together at the event twice, for Techorama Belgium and The Netherlands, the design needed to be very well thought out. All considering the constraints of both venues, such as the available space.

During the preliminary brainstorming sessions, the idea of incorporating an Escape Room emerged. That would definitively bring an epic experience to the visitors of our booth, if well executed. But, as mentioned we really wanted to step it up a notch compared to previous years in how well our booth spoke to our target audience. The challenge lay in seamlessly intertwining topics related to DevOps, Azure and GitHub, as well as our company culture and values into an engaging game experience.

Tapping into the Power of the Collective

"A clean whiteboard, a few markers and a lot of creativity"

Within the spacious confines of the airplane, we realized there was enough room to host an Escape Room. At first, our idea was simple: let people at the conference engage in some puzzles or challenges centered around DevOps and Azure-related topics. The possibility of building an actual Escape Room wasn't immediately evident. Taking this abstract idea and turning it into something tangible was difficult.

We believe in Sharing Knowledge. The more you share it, the more it grows. With bi-weekly sessions of Knowledge Exchange, we ensure a flow of knowledge from country to country, unit to unit and employee to employee.

We organized a brainstorming session during our knowledge-sharing evening and about 10 people turned up, including folks from our USA office. As we started sketching the Escape Room concept, the room buzzed with ideas. Rapidly jotting them down and grouping them on a whiteboard, this gave us clarity on the Escape Room project's direction. It became apparent that our puzzles would require answers in numerical form, which would result in a final solution allowing participants to "escape."

Additionally, we believed each puzzle should be meaningful. Our office is adorned with objects, or "artifacts" as we chose to call them, that symbolize achievements and milestones of our company. For each artifact, we envisioned a corresponding puzzle. By the end of this one-hour session, we had a foundational blueprint for our Escape Room, complete with tasks like establishing nine artifact-inspired puzzles and designing an input device.

Refining the Escape Room concept

Some people in the session were very enthusiastic about this Escape Room and decided to continue in a follow-up session to make the actionable things more concrete.
We managed to think of at least 6 Artifacts with a corresponding question and answer. The other 3 artifacts were gradually added to the 9 questions. We thought about ensuring that the first people who crack the code should not be able to share that code with others. Moreover, a team should not get the same questions as other teams.
We considered rotating all 9 puzzles, making them appear in random order. These are great ideas, but to bring them to life, the most viable solution was to craft a digital experience, complete with an input device and a screen to display the Escape Room's status. Then Thijs suggested:

"I can make an app ... ?!" - Thijs Limmen

Little realizing what he would be volunteering for. We began designing mock-ups for various screens the Escape Room would require: team input, an introduction video, question and answer input, result screens, and a leaderboard.
Also during this session, Thijs took the initiative to bootstrap a Flutter App, crafting an input screen for questions and answers. This progress boosted our confidence in building the app.

One Month to build an airplane & Escape Room

After the brainstorm session there was just one month to build the entire thing. Olaf already started in his workshop building the base fuselage of the airplane. It consists of five one-meter-wide elements that are fully demountable, kind of like an IKEA fuselage. In total it is five meters wide and two meters deep.

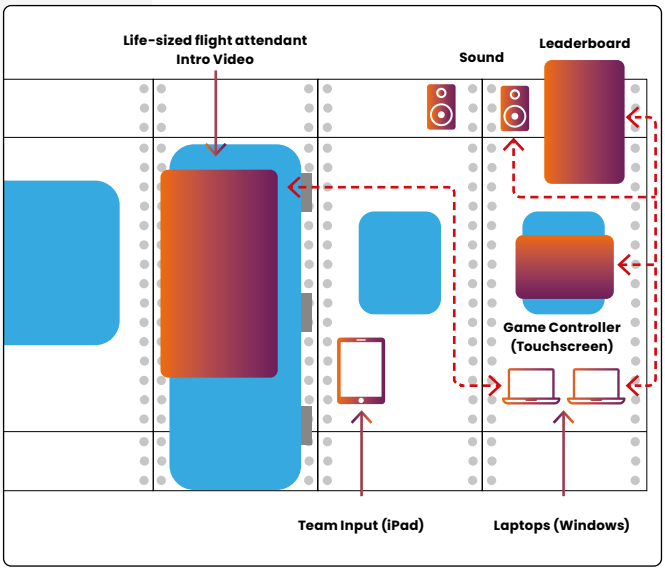
As it started off as a wood frame, one can imagine it took some effort to make it look more like an actual airplane fuselage. By emphasizing the inner truss structure and making it look like metal, by painting the outside of the plane white and adding hundreds of fake rivets, a door, windows, and so forth, it really started to look like an actual airplane fuselage. But new. Too new.

Then came the tedious task to make it look like it has been in the jungle for some time. It had to look abandoned, overtaken by nature. Inside and outside. We used all sorts of techniques to resemble things like rust and algae buildup. Different types of paint, coffee grounds, rust effect paint, sanding, leaves, spray paints, the grass powder a typical model train builder is common with, all the layers were building up to something that was really like our imagination.

All the while we were iteratively building on the digital Escape Room game. The physical booth and Escape Room appeared and next to it was Thijs, building the digital Escape Room game.

Behind the Scenes: The Tech Powering the Experience

The illustration provides an overview of the hardware components orchestrating the Escape Room experience. Inside the airplane, tucked within a cabinet, are two laptops: one drives the Escape Room's back-end, manages the Game Controller (touchscreen), and delivers all the associated sounds and videos. The other laptop handles the display for the Leaderboard and the Introduction video. Positioned outside the airplane is an iPad where participants input two or three player names along with a team name. Once submitted, there\'s a brief moment of anticipation allowing the team to prepare within the Escape Room.

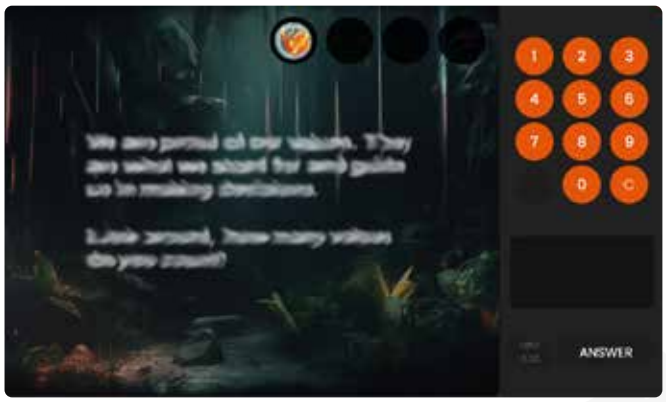


Shortly after, a life-sized video of a flight attendant comes alive on a massive screen, delivering a narrative and outlining the Escape Room's rules. When the video is done playing, the Escape Room starts playing automatically after a 10-second countdown. The game interface appears on the touchscreen and the huge display transforms into a ticking timer, initially set to four minutes. Participants inside the Escape Room are drawn into an atmospheric experience: the relentless tick-tock of the clock, mood-setting background music, and, as the last 30 seconds approach, intensified sounds raising the stakes. Encouraging audio cues chime in when they're on the right track, ensuring a deeply immersive experience. As teams either conquer or are defeated by the Escape Room, their performance is immortalized on the Leaderboard bolted on top of the airplane, in plain view for conference attendees to measure up against.

The Power of Generative AI

Several months prior to the Escape Room's construction, Thijs delved into the world of generative AI techniques, one of which was Midjourney (www.midjourney.com). This tool could be prompted to visualize any concept, which we found intriguing.

We tasked it with conceptualizing an Escape Room tablet interface incorporating a numeric input, laying the groundwork for our input device's aesthetic. However, it initially lacked the desired Jungle ambiance. Thus, we prompted Midjourney with an "abandoned jungle" theme using a reference image. By fusing these two designs, we achieved an input device that not only suited our needs but also embraced the jungle motif.



Imagine An Escape Room tablet app interface that displays a challenging question on the screen. Below the question, include a numeric keypad for users to input their answers --ar 16:9

Imagine <STOCKPHOTO_IMAGE> abandoned jungle theme wallpaper, flat, dark, grunge --ar 16:9

In line with our vision, we decided to feature our esteemed colleague, Natascha, as the flight attendant introducing the game. Filmed against a green screen, we envisioned her poised in front of an airplane door that embodied the abandoned jungle theme. With Midjourney's help, we secured the perfect backdrop.

Imagine airplane open door in abandoned jungle theme, cool color palette, medium-full shot --ar 2:3

In terms of showcasing scores, the exterior of the Escape Room needed a leaderboard that synced with the jungle theme. The concept of a flight departure board displaying team scores came to mind. When Midjourney initially presented multiple side views, we adjusted our approach, emphasizing a medium-full shot and a frontal perspective. The result? A captivating flight departure board that seamlessly integrated with our theme.

Imagine flight departure board in abandoned jungle theme, cool color palette, medium-full shot, front::2 view --ar 2:3

Technical solution of the Escape Room

Our technical choice leaned towards the Flutter framework for the escape room app, a decision influenced by Thijs's familiarity and positive experiences with it. Flutter's appeal lies in its ability to craft an app once and then compile it natively for diverse operating systems, including the web. This flexibility meant we could develop a singular app without pre-committing to a specific platform. In the end, our solution was deployed on both an iPad and a Windows laptop. Additionally, Flutter boasts exceptional performance and a developer-friendly environment. While our Escape Room is built using a single app solution, it's worth noting that four instances of this app run concurrently across multiple devices to show the various screens.

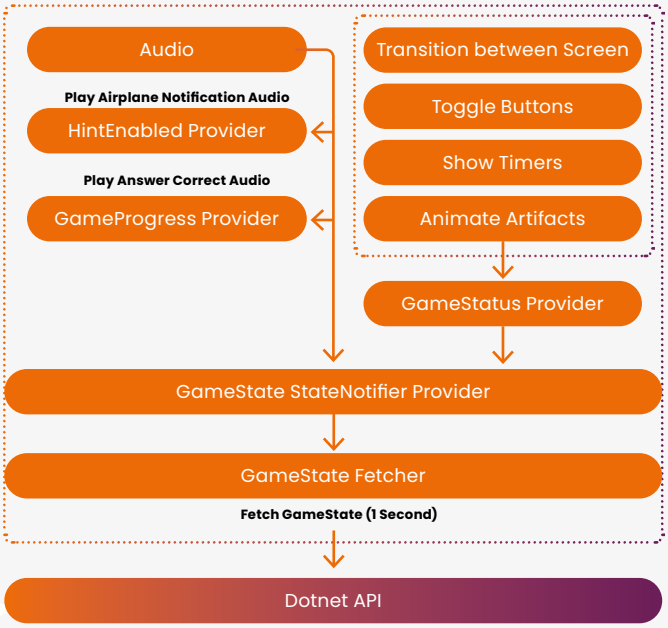
For the backend, we gravitated towards a Dotnet API. This encompasses various functions, from team submissions and game state retrievals to answering questions and

accessing the leaderboard. The game logic is handled in-memory, and when a game is completed, data gets stored in a MongoDB database. This backend solution is locally hosted on one of the Windows laptops and is made accessible externally through an ngrok reverse proxy.

Crafting a Reactive Frontend Game Experience

Our frontend solution is structured using the 'RiverPod' package, a notable library dedicated to Reactive Caching and Data binding. Within RiverPod, we utilize the State-NotifierProvider, and for our application, we've created one specifically for the GameState. To ensure real-time responsiveness, the GameState is fetched from the backend every second and subsequently cached within the GameStateStateNotifierProvider. Several Providers are then synchronized with this state. A prime example is the GameStatusProvider, which facilitates reactions to state changes---be it playing audio, triggering a video to start, initiating animations, or switching between screens. RiverPod's architecture made it seamless to manage state transitions throughout our Escape Room game.

Furthermore, we integrated packages designed for audio and video playback.



Dotnet API powering our Escape Room's Logic

Our backend is anchored by a straightforward Dotnet API. This houses the 'GameController', equipped with a fetch GameState endpoint, submitting answers, inserting teams, and other vital functionalities. Moreover, it contains a LeaderboardController dedicated to fetching the leaderboard. A central component is the ActiveEscapeRoom class, encapsulating the state and mechanics of a live Escape Room session. Simplifying the process was key, as we wanted the support and management of the code to be straightforward and due to the time constraint of one month to build the Escape Room. Roughly 25 integration tests were devised, mimicking the frontend application's interaction with the backend, ensuring various scenarios like automatic game termination after 4 minutes, hint activation, game success metrics, and more. For the integration tests we used '**Microsoft.AspNetCore.Mvc.Testing**' library

We incorporated the '**StateMachine**' library to regulate state transitions and preventing any unanticipated shifts in state. Notably, we opted against unit tests since the Escape Room's implementation was in a constant state of flux. This strategy allowed for frequent modifications to the Escape Room's internal logic without compromising the game's integrity or introducing glitches.

Automating App Builds with GitHub Pipelines

To streamline our development process for the Escape Room apps, we established a GitHub pipeline. This pipeline was designed to build the various Flutter apps required, including those for IOS, Windows, and Web platforms.

Importantly, the pipeline would be triggered with every tagged version of the app, such as "v1.1.0-release1."
To initiate this, one would simply use the commands:

```
git tag v1.1.0
git push --tags
```

Link to the GitHub source code: https://github.com/ThijSlim/Techorama-2023-XpiritXebia-Escape-Room
(The source code of the Escape Room will be made available on GitHub after the Techorama Netherlands conference.)

First official flight at Techorama Belgium

The Escape Room in the abandoned airplane took its inaugural run at Belgium's Techorama conference.
Just 5 days prior, we finalized the airplane, readying it for the journey to Belgium. In the following days, we put the final touches on the Escape Room, ensuring it was set to welcome attendees. One day ahead of the conference, the airplane was assembled, and by early afternoon, we initiated test runs. By the close of the day, all systems were GO for the two-day Techorama Belgium event.

The subsequent day, the first team eagerly stepped up to try their hand at the Escape Room. Their experience? Less than triumphant. They managed to solve only one puzzle and didn't escape. This pattern continued with several following teams. We deduced that some puzzles were overly tricky, certain questions muddled the participants, and the hints didn't quite cut it. Responding swiftly, we spent the morning refining our setup, even extending the game time from 3 to 4 minutes. As these changes took effect, more teams began to succeed and from that time we stopped making further adjustments. As the day wrapped up, Thijs had to depart from the conference due to personal commitments. Was this a good idea? Let's find out.

Day 2 dawned with a hiccup. Thijs received a call from colleagues at the conference site: the Escape Room was not starting a new game. The issue? The conference network wasn't cooperating with our ngrok reverse proxy. A quick switch to a different Wi-Fi network sorted this out. But another problem arose: the Escape Room would sporadically stop working. A simple backend restart fixed this temporarily. After the conference we pinpointed the cause was the team's queuing mechanism was causing a memory leak. Despite these hitches, the Escape Room was largely operational. It was awesome to see the Escape Room in action at the conference. It was fun to see people playing the Escape Room and walking out with a big smile on their face or a bit sad, because they didn't escape.

An improved Escape Room for Techorama Netherlands

From our experience running the Escape Room in Belgium, we gained many insights. With an upcoming event in the Netherlands, our aim is for the Escape Room to be rock solid right from the start and to maintain stability over the conference's two-day span. One of our immediate resolutions is to establish a local network, negating any reliance on the often-unpredictable conference WiFi and its accompanying policies. For added assurance, we're considering bringing more robust laptops, particularly for backend operations. This preparation ensures any potential memory or CPU hitches are addressed more seamlessly. Additionally, we've taken measures such as performance and load testing, and rectified the previously mentioned memory leak.

Ahead of the Netherlands conference, we also took the time for internal testing of the Escape Room. We couldn't fit this phase into our schedule before Techorama Belgium. The Leaderboard saw refinements; not only does it now reflect the progress of teams that couldn't escape, but it also showcases failed games. This change was mainly done because at Techorama Belgium we had an empty leaderboard for the initial 4-5 hours and we also think it's good to honor teams that didn't make it. Furthermore, we've enhanced the user journey, streamlining the process of submitting player and team names to start the Escape Room. Lastly, post the introductory video, the game now automatically starts after 10 seconds. This is a marked change from the Belgium setup, where teams cleverly used more time to familiarize themselves with the Escape Room's environment.

From Vision to Reality: Our Epic Escape Room Adventure

Building an airplane and an Epic Escape Room in such a short amount of time has been awesome. Reflecting on it, we believe this might rank as one of the most epic experiences we've ever had. </>

P.S. If you're reading this during the Techorama Netherlands conference, come and stop by the abandoned airplane and play our Escape Room.

Reflections of a DevOpsologist

At heart, I'm a developer. I love to sling code and be part of a team that slings code. But I'm also fascinated by 'people' - and the intersection of culture and tech is what has drawn me to DevOps. As I reflect over my journey, I see a lot of people who invested in me. Without them I would not be where I am today. Quite literally - I am an immigrant to the US and my career brought me here. I've also worked hard when opportunity presented itself. I've been mentored and have mentored others. I have learned some things along the way that I hope I can communicate through my story.

Author Colin Dembovsky (GitHub)

But wait - before I start: what is a DevOpsologist? I forget where I heard the term from (it's not mine) but I love the sentiment. It comes from DevOps and the suffix "-ology" (the study of something, a branch of learning). DevOps is continually evolving and changing, and I don't think we'll ever "arrive". Calling myself a DevOpsologist reminds me that there is always more to learn!

The Story

The Journey Begins

It was summer (at least in the Southern hemisphere) of 2004. I had just moved from Johannesburg to East London, South Africa where I joined a team of about 20 developers for a financial services company. I had been hired on as a senior developer - and little did I know that the next six years would come to define so much of my career.

I was now about three months into the job. My previous job hadn't been anywhere close to anything from Microsoft. It was all C++, CORBA[1] and Linux. Now here I was, a senior developer at a team using SQL Server and Web services built on .NET Framework 1.1. I had used CVS for source control before, and the new team was using WinCVS. It wasn't pretty. There was almost no process in place - we deployed by using Visual Studio's "right-click Publish" feature and fat-fingering was so common it was expected.

I remember thinking to myself, "I may not be the most experienced .NET developer, but there must be a better way to manage how we deliver code." So I opened up a browser and used my favorite search engine, WebCrawler, to see if I could find anything better.

And I did. I found a tool that was so new it was still in beta. The installation took about a week - at one point the install failed and I couldn't recover. In fact, the failure was so bad I ended up formatting the entire machine and starting again. But I persisted - and finally stood up a shiny new instance of Team Foundation Server (TFS) 2005 beta 2.

The Glorious TFS Days

So began my journey into DevOps. Except that the term DevOps hadn't been coined yet. It was still "Application Lifecycle Management" (ALM). I don't even think I heard that term until around 2008. But even if I didn't know what it was called, I was doing it. Mostly by instinct.

We started by getting all of our code into TFS. We even started using automated builds with MSBuild! We could at least build from a known source of truth, rather than hoping what we had on our dev machines was the latest code.

Our next phase was adopting unit testing. I remember some very heated debates with the team. "We have too much code and our coverage will be practically zero!" was a common sentiment. I managed to convince the team that 0.2% code coverage is better than 0%, so we started by just ensuring that if we touched a method (or added one) that we would only deploy if we had a unit test for that changed code. Before long, we were in the 60% range for code coverage. Small, incremental changes added up having a large impact over time.

We also started using Work Item Tracking. We all underwent Prince2 training (it's a flavor of Waterfall from the British Government) and customized our templates, work items and reports to match. Back then we had to create our own Release Management tool since there wasn't one in TFS till years later.

Jump to Consulting

In 2008, I attended TechEd Africa in Durban, South Africa. This was my first large tech conference experience. I went to a talk by Chris Menegay, who ran a small consulting firm in Dallas, Texas called Notion Solutions. Chris had an ALM talk that featured TFS. I knew I wanted to move into consulting at some stage, but until that point I had no idea what to consult in! Hearing Chris talk about his team of ALM consultants, I knew that was what I wanted to do. However, my first child had just been born, and I didn't think it was the right time to leave a job with a steady salary.

I attended TechEd Africa in 2009 - and Chris was a guest speaker again! This time I was ready for a change - so after his talk, I had a 5 minute conversation with him. I remember

[1] https://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture

asking him if he did any work in South Africa, and he replied he could send a consultant over (later I found out he was thinking of Donovan Brown who worked for Chris at that stage).
"No, I'm looking to get into consulting," I responded. Chris and I had a chat with a key Microsoft figure - and someone I owe a debt to for all the help in my early days - Ahmed Salijee, who headed up Visual Studio sales for Microsoft in South Africa.

For some reason, Chris took a chance on me - and a few months later I started the South Africa branch of Notion Solutions! Chris put a few thousand dollars in so I had a steady salary, but I did everything. I was calling. I was delivering. I was invoicing - I learned so much during that time. Fortunately, I had the collective minds of Notion Solutions backing me, so I started confidently, even though I didn't really know what I was doing. That was a good way to grow.

How did I get here?
In September 2010 Chris flew me out to a Notion Solutions gathering in Irving, Texas. It was a rare occasion to have all the Notion Consultants in a single place. I remember feeling awed. There I was in the same room as some of my "ALM heroes" such as Chris Menegay, Dave McKinstry, Abel Wang, Donovan Brown, Steve St. Jean and Ed Blankenship. "How did I get here?" I wondered - but I was determined that I would learn all I could from these folks and wouldn't take my good fortune for granted!

Most of the Notion Team were Microsoft Most Valuable Professionals (MVPs) and I was inspired to attain that award too. So I started my blog - Colin's ALM Corner[2] which is still going today. In 2011, I was awarded my first MVP award. In 2012, I got to attend my first MVP Summit in Redmond, WA. There I met some folks that I am still connected to this day - leaders in the ALM community such as Marcel de Vries, René van Osnabrugge, Pieter Gheysens, Brian Randell, Nino Loje, Mickey Gousset, Esteban Garcia, Martin Hinshelwood and many others - including Steven Borg.

Moving to America
Steve and I connected really well - and eventually he offered me a position at his company Northwest Cadence. Northwest Cadence was based in Seattle and was also a small ALM consultancy. Northwest Cadence persevered through all the legal processes to move me and my family over to Seattle in 2016, and I've been in the US since.

Steve inspired me - he is an excellent communicator who knows something about everything. And his mastery of lean processes and agile was amazing. I remember thinking, "When I grow up, I want to be just like Steve!" I learned so much during my North-west Cadence days, and still think in terms of flow, efficiency and queuing theory.

In 2018, Steve merged his company into Chicago-based 10th Magnitude. 10th Magnitude was doing some great Azure work, but was finding more and more customers wanted to modernize their software processes as they migrated their data centers to the cloud. The folks from Northwest Cadence brought a wealth of process consulting and DevOps to 10th Magnitude, so it was a really good fit.

Moving into... sales?
Soon after joining 10th Magnitude, there was an opening for a Solution Architect. I looked at the position and chatted to a few folks, and discovered it was a technical sales role. "I'm not a salesman, I'm a consultant!" was my initial reaction. However, I negotiated with the leadership and took the role on the basis that I would sell for 30% of the time, and consult for the other 70%.

That never turned out to be the case. I ended up selling far more than consulting. But one day I had an epiphany: I love to solve complex problems. And I was doing that in my sales process! I would meet with customers, and spend time to understand their environments, people and challenges. I then crafted services and deals that we would deliver to our customers to help them achieve their goals. While I was now in a sales role, I was able to do what I love doing - use tech and cultural engineering to solve problems.

What is this Git thing?
I remember when TFS introduced Git repositories around 2013. I couldn't see the appeal - who would use a source control system that let you overwrite history? However, after spending some time with it I started to see the light - so much so that I did a talk at VSLive for a couple of years where I hypothesized that you can't really do modern development if you're **not** on Git!

And then - Microsoft purchased GitHub in 2018. So - reluctantly - I started to figure out how to use the platform. I preferred Team Foundation Server - which had changed names a couple of times and is now Azure DevOps. That is, until GitHub released GitHub Advanced Security (GHAS).

AppSec is the Future
I have a development background - Security was always the team that "prevented you going to prod". I didn't speak security, and I had never met a security professional that spoke developer.

[2] https://colinsalmcorner.com

But GHAS was different. I instinctively guessed that this was a tool that I wanted to align with. Over time, I was able to verbalize this instinctive feeling - it's simply **security tools for developers**. I recognized that this was a game-changer.

At this stage I was the DevOps Practice lead at 10th Magnitude, and I created one of the first GHAS partner offerings. We had a 2-week GHAS Adoption service and were able to help onboard a few companies to GHAS in the early days.

Moving to GitHub

I also got to deliver some GitHub/10th Magnitude Roadshows. I met a few Hubbers during that time - including Kevin Alwell, a Solution Engineer on the east coast. In 2021, I applied for a Solution Engineering position at GitHub - and about two days later, Kevin called me out of the blue. After catching up, he told me there was a Solution Engineer role he thought I would be good for... and the rest, as they say, is history.

Reflections

I've had an incredible journey - and still have lots to look forward to! The advent of generative AI through ChatGPT and GitHub Copilot is just beginning to revolutionize development as we know it. If software has eaten the world, it's now AI's turn!

Since I am a self-confessed DevOpsologist, I proclaim to be constantly learning. So what have I learned over the years working in DevOps?

Find your inspiration outside of your career

My faith and my family are the core of my identity. I love being an SE and a technologist - but that's what I do, not really who I am. This has been vital to handling pressure and hard times - when work sucks (as it inevitably will be from time to time), I don't feel that **I suck**. I've found that, ironically, living for something _other_ than work and technology has led to more fulfillment in work and tech! So find something that you can be passionate about that isn't your work - and your work will actually improve!

People matter

I love to sling code. I love being a geek. But no matter how technically proficient I am, and no matter how amazing the tech is, people are still the heart of DevOps. I see a lot of companies that fail not because they are not smart, and not because they have the wrong tools, but because they don't prioritize people and culture.

One of my favorite Laws is Conway's Law. Conway (a programmer) introduced an idea in 1967 that showed a "homomorphic force" between the communication structure of a company and the architectures it produced. In other words, the culture plays a vital role in shaping the technology.

If you haven't yet read Team Topologies[4] do yourself a favor. Stop debating if you should implement microservices and spend some cycles on designing your **Teams**. In other words, people matter - so align with that force so that you're not fighting it all the time.

I have had to learn (and still am learning) that _how_ you say something is just as important as **what** you say. This was something I had to learn as a consultant - and I still have to work on it every day. I can come off as cutting and dismissive - a byproduct of my wiring to see to the heart of a problem very quickly. But I have to constantly think about how I communicate what I see. Because people matter.

Stay teachable

A corollary of the above is that **you matter**. And if you matter, then you're worth investing in. One of the best ways to invest in yourself is to ask for feedback (or consider unsolicited feedback carefully). We all have blind spots, biases and areas we can grow in. If you are not able to admit when you are wrong - and learn and grow - then you're going to cap your potential.

Managers, peers and customers have all at some time or other given me feedback about something. Each time I try to figure out what I can learn from that feedback. At times, I have "chewed the meat and spat out the bones". Not all feedback is always correct - so I try to find the things that I can learn and grow from - and ignore the rest.

Keep learning

This is a core strength of mine from Gallup's CliftonStrenths[5]. So learning comes easily to me - that's not the case for everyone. But I believe that everyone should always be learning. And it's close to being a requirement in DevOps given the pace of the software industry.

I could have ignored GitHub Advanced Security - after all, I'm not a security professional. However, I applied myself to learn about it - and it's been key to my success in the past few years.

Sometimes, you'll need to just knuckle down and learn about something even if it's not "in your lane" - you'll be surprised at what might happen!

---

[3] https://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture
[4] https://teamtopologies.com/
[5] https://www.gallup.com/cliftonstrengths/en/strengthsfinder.aspx

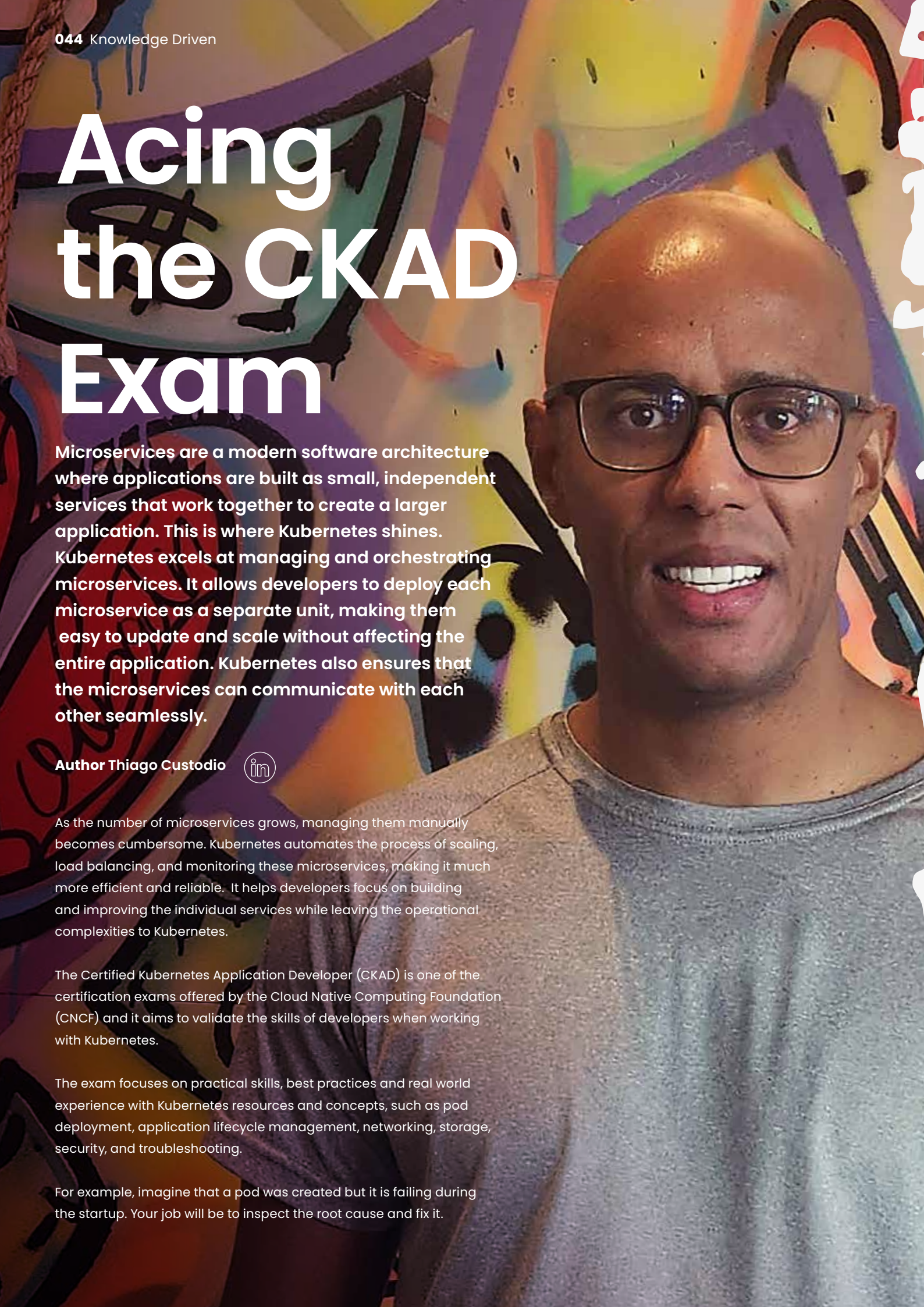
Take calculated risks

I didn't know if consulting would be a good long-term choice for my career. I didn't know if moving to the US would be good for my family. I didn't know if moving into sales would be something I could do long-term. I approached each decision as rationally as I could, seeking input from friends, family, peers and managers as appropriate. But I never let myself get into "analysis paralysis". At some point, I had to take some calculated risks. Thankfully, I've had a good run, even when there was uncertainty.

In closing

Being in DevOps has been incredibly fulfilling - from the work I've been able to do, to the people I've met, to the ways I've grown and developed as a person. Every day I count myself fortunate to have a job I love - and to be in an industry that's continuously changing and evolving. I hope that my story and some of my learnings can inspire you to keep growing - and hopefully, I get to hear your story some day. </>

Acing the CKAD Exam



Microservices are a modern software architecture where applications are built as small, independent services that work together to create a larger application. This is where Kubernetes shines. Kubernetes excels at managing and orchestrating microservices. It allows developers to deploy each microservice as a separate unit, making them easy to update and scale without affecting the entire application. Kubernetes also ensures that the microservices can communicate with each other seamlessly.

Author Thiago Custodio

As the number of microservices grows, managing them manually becomes cumbersome. Kubernetes automates the process of scaling, load balancing, and monitoring these microservices, making it much more efficient and reliable. It helps developers focus on building and improving the individual services while leaving the operational complexities to Kubernetes.

The Certified Kubernetes Application Developer (CKAD) is one of the certification exams offered by the Cloud Native Computing Foundation (CNCF) and it aims to validate the skills of developers when working with Kubernetes.

The exam focuses on practical skills, best practices and real world experience with Kubernetes resources and concepts, such as pod deployment, application lifecycle management, networking, storage, security, and troubleshooting.

For example, imagine that a pod was created but it is failing during the startup. Your job will be to inspect the root cause and fix it.

About the Exam

Before starting the exam, you will need to download a specific browser provided by PSI (a company leader in the assessment industry). In my first attempt to do it, I was trying to install it using the corporate laptop, but it had some custom policies which were conflicting with the browser. Make sure your computer/laptop is compliant with the PSI browser or you won't be able to launch the exam.

The exam is a collection of 15-20 exercises, and you have two hours to complete it. It is 100% hands-on, and you need to score 66% or above to earn the certification. If you have experience with Kubernetes, you probably just need to review what's new or what has been deprecated in the current Kubernetes version. Also, it is worth it to review a few commands that you probably don't use often in your daily activities.

TIP #1 - How to study?

My first engagement when I started at Xebia | Xpirit was a large migration from AWS to Azure. I acquired some knowledge on Kubernetes while working on this project but my peers were already familiar with the ecosystem. For someone with almost no knowledge on Kubernetes to suddenly get exposed to Helm and advanced Kubernetes concepts, I realized there was a gap in my knowledge. I decided to dive deep and learn Kubernetes for real. After some research, I bought the book **Kubernetes in Action** from Manning and some training courses from KodeKloud[1] and Linuxtips[2] (the second one is available in Portuguese only).

Although you don't need to purchase the training courses, I thought study through them would accelerate my learning process. Besides the great explanations, both platforms offer real labs where you will connect to their Kubernetes cluster and perform the exercises. This experience is similar to the real exam, so I recommend them.

Another useful resource I used was Docker Desktop. After installing it, you can enable Kubernetes and it will provision a local cluster for you:

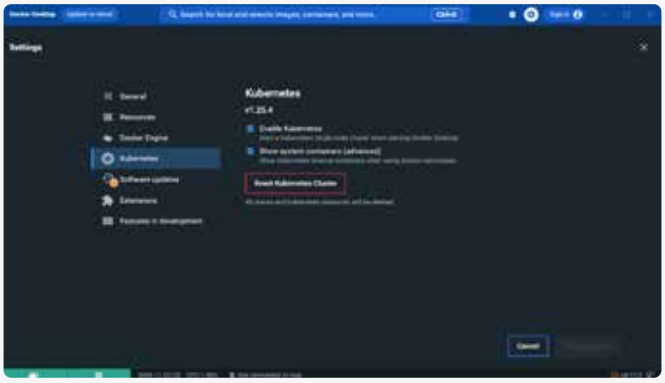


Figure 1: Enable Kubernetes using Docker Desktop

You can also use Minikube or Kind in order to create a local cluster. If you have a valid subscription you can create a managed Kubernetes cluster on Azure, but make sure you delete all the resources after using it, or it will consume your Azure credits. You can get more information about how to create a local Kubernetes cluster in the following links:

- https://kind.sigs.k8s.io
- https://docs.docker.com/desktop/kubernetes

Tip #2 - Be quick or be dead.

Every little second counts, so it is important to learn how to solve the problems as fast as possible and get familiar with techniques which will save you time. A few tips I can give on this:

2.1 - Use the dry-run option.

When creating resources in Kubernetes, you can either type the entire yaml file, or use the kubectl to generate a yaml file with the parameters you specify. For example:

```
kubectl run mypod --image=nginx -n test --dry-run=client -o yaml > generated.yaml
```

The previous command will produce the following yaml file:

```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: mypod
  name: mypod
  namespace: test
spec:
  containers:
  - image: nginx
    name: mypod
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

I don't know how fast you can type, but in my case kubectl saved me a lot of time. It worths to mention that the parameter `dry-run=client` will not create any resource in Kubernetes, so it is useful to validate before deploying resources and to save you some time avoiding the typing of the entire yaml file.

[1] https://kodekloud.com
[2] https://linuxtips.com.br

2.2 - Edit an existing resource / get the yaml of an existing resource

Some exercises will ask you to change a resource which already exists. You can either use `kubectl edit` or `kubectl get` with the option to get the output in yaml format and redirect it to a file:

```
kubectl get pod mypod -o yaml > existing.yaml
```

The previous command will download all the details and specifications of the selected pod ('mypod') and create the file 'existing.yaml' with its content. You can now open the 'existing.yaml' and perform the changes requested. After that, you can use kubectl replace which will delete and create the resource:

```
kubectl replace --force -f existing.yaml --grace-period=0
```

PS: The grace period parameter is used to specify the period of time in seconds given to the old resource to terminate gracefully.

Tip #3 - Master your text editor

You can use other text editor rather than Vi. I did my exam using Vi, but even if you opt to use a different one, it is important to learn how to be productive on it. Learning how to select and replace multiple instances of a given text, add or remove indentation on multiple lines, cut and paste multiple lines respecting the indentation are some of the tips I can give on this topic.

Tip #4 - Learn how to search for what you need in the official doc.

During the exam, you are free to open the Kubernetes Documentation[3]. Sometimes it will contain exactly what you need, but in other cases, it will give you an explanation but will not give you a concrete example. For those cases, you can use `kubectl explain` and navigate through the documentation.

For example, let's say you will need to add an environment variable for a pod, but you can't remember where to place them in the yaml file. You can use kubectl explain with the recursive option:

```
kubectl explain pod --recursive
```

This will give you a lot of information. All you must do is navigate through hierarchy:

```
pod -> spec -> containers -> env
kubectl explain pod.spec.containers.env --recursive
```

This will give you the structure / valid parameters for that section:

```
FIELDS:
   name <string>
   value        <string>
   valueFrom    <Object>
      configMapKeyRef   <Object>
         key    <string>
         name   <string>
         optional       <boolean>
      fieldRef  <Object>
         apiVersion     <string>
         fieldPath      <string>
      resourceFieldRef  <Object>
         containerName  <string>
         divisor        <string>
         resource       <string>
      secretKeyRef      <Object>
         key    <string>
         name   <string>
         optional       <boolean>
```

Tip #5 - Train a lot!

It is hard to be ready for a topic such as Kubernetes as it is an extensive topic. My recommendation is for you to try the previous tips and explore the possibilities. For example, create a pod with a single environment variable. After that, create another one, exposing Config Maps as environment variables. Then another one, but this time exposing Secrets as environment variables. Lastly, learn how to select a subset of items from secrets / config maps and expose them using a different name than what is in the Config Map / Secret.

I use this technique in many areas of my life. It is a 'drill' where you exercise something on and on with small variations. Do it until it becomes natural due to the amount of repetition you did during the drills.

You can get more information about the exam at the official website: https://bit.ly/CKAD-EXAM. </>

Conclusion

Kubernetes is an advanced topic. I studied for months in order to succeed on this exam. You must not only master the concepts, but be able to solve the problems within the exam duration. I've shared useful tips which will save you time even in your daily Kubernetes activities.

Feel free to connect with me on Linkedin or Twitter. I will be interested to know your thoughts on the tips I've shared in here. If you have plans to take the exam, I wish you good luck! Stay Hungry, stay foolish!

[3] https://kubernetes.io/docs

Fuzzing in C#

In our previous issue of Xpirit Magazine, we delved into the realm of Mutation Testing in C#, an accessible strategy for identifying weaknesses in your automated tests, thereby enhancing the overall quality and stability of your software. If you missed that, don't hesitate to grab your free copy of Xpirit Magazine #14!

Today, we're exploring another technique that can significantly elevate your software's quality: Fuzzing. At its core, fuzzing revolves around a straightforward concept: Supply a program with invalid, random, or unexpected input until it encounters a crash.

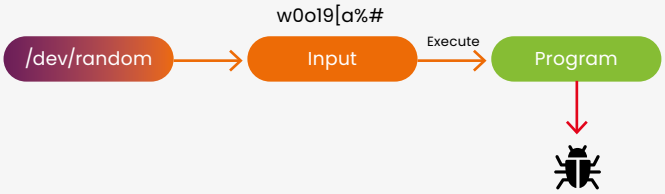
Author Michael Contento

Understanding the basics

Imagine you've developed a program designed to process JPEG images. Now, picture loading a PNG image, a PDF file, a hefty 200MB PowerPoint presentation, or even a file filled with random gibberish. What should the program do? Well, that depends on your program, but a controlled exit with an informative error message (such as "**the file you provided is not a valid JPEG image**") seems reasonable. Depending on the file-check implementation, you might reject the PDF file, but the PNG image might sneak through your initial validation since it's a valid image format. But how does your image decoder, the component responsible for reading and decoding the JPEG binary stream, respond to the bits and bytes of a PNG file? Does it gracefully continue or crash catastrophically? It might survive if it can't understand the bytes that constitute the file header. However, what if we substitute the PNG file with a corrupted JPEG file? One with a valid file header but random data thereafter?

The potential for errors is vast. While rigorous software development practices like comprehensive automated testing, and even Test Driven Development (TDD), combined with consideration for edge cases, can guard against many issues, there's always the possibility of unforeseen errors.

This is where fuzzing comes into play, automating the process described above. You specify the program you want to test, and the fuzzing tool hurls randomly generated files at it until it discovers something that triggers a crash. These crash-inducing files can then be manually examined, analyzed, evaluated, and used to rectify the root causes of these crashes. Ultimately, this boosts your product's quality, particularly when considering vulnerabilities like memory corruption or exploitable buffer overflows.



Challenges with pure random data

Now that we've covered the fundamental principle, let's delve into the first step: generating the files to feed into our program and, more critically, where to obtain them.

The challenge here lies in the fact that we **(a)** cannot supply the files ourselves, as part of the fuzzing process is uncovering unknown problems, and **(b)** pure random data is not ideal. Indeed, generating pure random data is straightforward; we could simply read from a random source like `/dev/random`. However, does it truly assist us? More often than not, pure random data is just noise within the context of our program.

To illustrate, consider Apache Ant, a build automation tool that interprets build definitions from XML files.

```
<project default="dist">
  <target name="init">
    <mkdir dir="${build}"/>
  </target>
</project>
```

Replacing the XML file with random bytes would result in chaos:

```
lrha3wn5p0w3uz;54 p0a23rw3i 50a20 5a2y58a2p
y3wry3p285 q@P"uer9zparu9apur9qa3802 y5o2y
392r523a90wesu
```

Clearly, our random bytes bear no resemblance to valid input. While it's possible to generate a valid XML file with random bytes, the odds are slim, and we want fuzzing to yield actionable results within a reasonable timeframe. We cannot disregard the runtime of our fuzzing endeavor.

Since random bytes are vastly different from reasonable XML, it's highly likely that we'll repeatedly encounter the same input sanity checks. Is the input a valid XML file? No? Exit early. This cycle would continue, bypassing any business-related code.

However, if we shift our strategy from pure randomness to a slightly "mutated" approach (the attentive reader might notice the relation to Mutation Testing), we end up with something like this:

```
<project default="dist">
  <taWget name="init">
    <madir dir="2{build}"/@
  </tar?get>
</project>
```

Here, things get interesting. Our new input still resembles XML but with minor defects — defects subtle enough to permit entry into deeper parts of our program while still being defects. Consider that you've written code responsible for handling a specific XML node, so that `<mkdir dir="foo" />` creates a directory named `"foo"`.

Where do you register this new action? Perhaps in a global lookup table where you assign your action callback function to the name `mkdir`? Excellent. This central registration point simplifies action definition and registration.

But how does our program handle access to this global hashtable? Does it handle it gracefully when trying to retrieve the callback function for an unknown action name? Or does it produce an out-of-bounds error because no one has ever tested it with invalid or unknown action names?

This illustrates that slightly mutated data is far more effective than pure random data. With a single, minor mutation, our build definition was able to trigger an invalid action (`madir`, just a single-character error away from the valid `mkdir`).

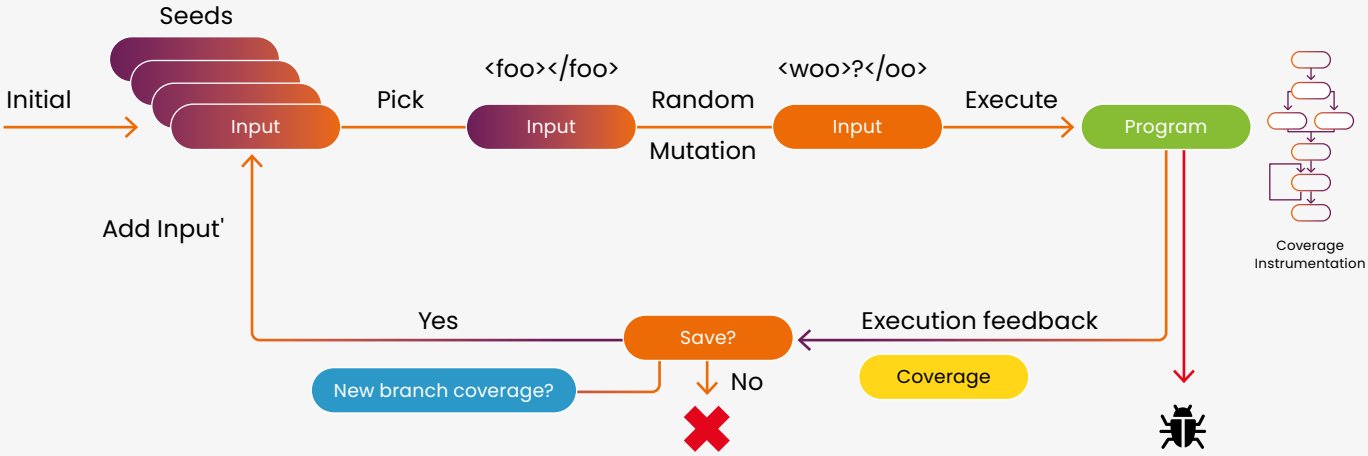
Tracking progress

We've established that mutated data surpasses pure randomness. But how do we gauge our progress? When do we decide that we've made enough attempts with mutated data? The theoretical space of possible function calls is seemingly infinite.

Wouldn't it be fantastic to somehow peer into the program we're testing, even briefly, to observe our progress? How deeply have we delved into the program with our mutated input?

This is precisely what **coverage-guided fuzzing** does. To implement this, we need a special build of our program with some instrumentation added. This instrumentation doesn't affect the program's behavior; it merely enables the fuzzing tool to monitor execution paths.

Now that we can track execution paths, we can assess our mutated input. This capability is crucial because it enables us to create a positive feedback loop, automatically steering us deeper into the program, thereby reaching more code with our malicious input. How does it work?



Let's examine the process:

1. Initialize and prepare some seed files, which can include entirely valid files like a functional Ant build XML file.
2. The fuzzing tool selects one of the seed files.
3. It applies a random mutation to the chosen file.
4. The mutated file is then passed to the program for testing.
5. **Did it crash?** If yes, we've discovered something, and we report the input file that caused the crash.
6. If no crash occurs, we examine the execution path (thanks to the instrumentation code).
7. **Did we traverse previously unexplored paths?** If yes, we add the input file to our collection of seed inputs.
8. If not, we can simply discard everything.

This loop repeats continuously, but Step 7 is the crucial one. Consider this for a moment: Every input file that leads to new execution paths within the program is added to our seed collection.

This process resembles evolution. A "first-generation" mutated input file might just "get its foot in the door". By reintroducing it into the seed collection, it has the opportunity to create a "second generation". This second-generation input file could progress further, perhaps fully infiltrating the program. We're essentially evolving through mutation and failure.

Automating the process

Up to this point, we've discussed theory. But how can we put this into practice? Do we need to build everything from scratch, or can we use existing tools? Enter **AFL**[1] and **SharpFuzz**[2]. Both open-source tools make fuzzing in C# a straightforward process.

AFL (American Fuzzy Lop) is considered the de-facto standard for fuzzing and enjoys widespread use. It has detected numerous significant software bugs in major applications such as OpenSSL, bash, Firefox, and SQLite. AFL is also widely used in academia, as academic fuzzers are often forks of it, and AFL is commonly used as a baseline to evaluate new techniques.

SharpFuzz extends the power of AFL to .NET. It's a lightweight library that facilitates the addition of the required instrumentation code (enabling AFL to work with .NET) and provides functions to simplify setup.

With both tools at your disposal, you don't need to concern yourself with the intricacies of fuzzing logic. Instead, you can focus on your code.

What to target

The final aspect to grasp is that fuzzing is versatile — you can target virtually anything. It doesn't have to be your entire application. Imagine you've created a Windows desktop app that can render HTML, and you want to use fuzzing to fortify this rendering process. But it's tucked away within layers of menus and buttons! Do you now need to "fuzzy-navigate" through the entire user interface?

Absolutely not. Sure, from AFL's perspective, it's simply an executable binary receiving mutated input files.

You can effortlessly create a small fuzzing harness, like this:

```
public class Program
{
  public static void Main(string[] args)
   {
      Fuzzer.OutOfProcess.Run(stream => {
          try {
              new HtmlParser().Parse(stream);
          }
          catch (InvalidOperationException) {
              // Whitelist known or "good" exceptions
          }
      });
  }
}
```

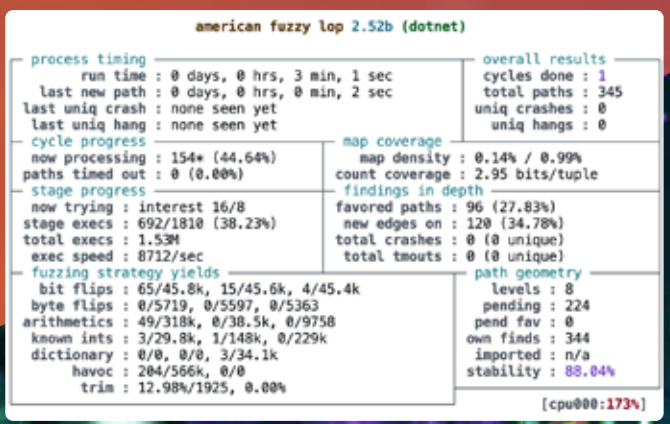
Here, `HtmlParser`[3] represents the HTML parsing library you wish to test. With SharpFuzz's assistance, creating a dedicated fuzzing harness is straightforward.

Notable here is the `InvalidOperationException` that we catch. From AFL's perspective, any program crash is flagged as "**erroneous behavior**" and tracked as a potential error. However, this isn't the case for `InvalidOperationException`. This exception serves as `HtmlParser`'s way to signal the caller that it encountered something it couldn't parse. To prevent AFL from flagging this as a false positive, we catch and whitelist this exception.

With the fuzzing harness in place, you only require a single seed input file, which can be a perfectly valid file like this concise HTML snippet:

```
<!DOCTYPE html><html><body><h1>h1</h1><p>p</p>
</body></html>
```

Now, you can direct AFL at your program and let it work its magic:



[1] https://github.com/google/AFL
[2] https://github.com/MetalNem/sharpfuzz
[3] https://github.com/AngleSharp/AngleSharp

While the output might seem overwhelming, it's a screenshot of the AFL Command Line Interface (CLI) output, an interactive Text User Interface (TUI) that allows you to monitor and follow progress in real-time. It provides vital information, including:

- Total run time (top left)
- Total unique paths discovered by AFL (upper right)
- Total program executions (middle left)
- Program executions per second (middle left)
- Mutation strategies applied by AFL (lower left)

This interface allows you to closely observe AFL's execution, and at some point, AFL might uncover a crash! You can then examine the file AFL generated:

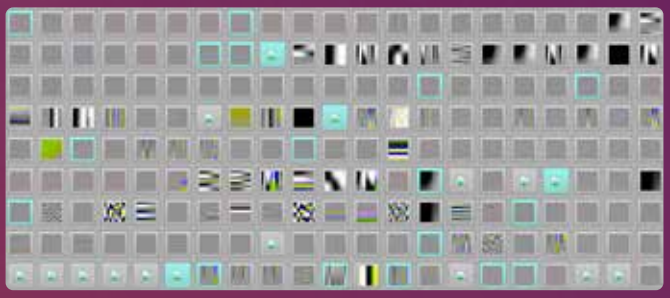
```
<svg><!DOCTYPE html><<template>html><desc>
<template>><p>p</p></body></html>
```

Success! We've found malicious input capable of crashing our `HtmlParser`. Now we can debug this issue, create a unit test to prevent future regressions, and apply standard development practices.

Creating images out of thin air

We've witnessed AFL mutate our simple HTML input sufficiently to trigger a crash in the `HtmlParser`. But how capable is AFL? How far can it stretch its abilities? In short, quite far!

I stumbled upon this intriguing article by Michal Zalewski[4], where he detailed how AFL was able to generate valid JPG files seemingly out of thin air:



Admittedly, these are not aesthetically pleasing images, but they are unquestionably **valid** JPG files. All created by AFL as it diligently mutated its way through a JPG decoding tool.

Conclusion

Fuzzing is an engaging adventure that can uncover bugs in your program. Importantly, it isn't limited to end-user applications. With a dedicated fuzzing harness, you can isolate individual functions or entire libraries and direct AFL to them.

However, due to its generative nature, it's essential to be mindful of runtime. Fuzzing will invariably be slower than unit tests. Therefore, consider it a complementary tool. Unit tests verify the known paths, but fuzzing excels at discovering the unknown paths! </>

[4] https://lcamtuf.blogspot.com/2014/11/pulling-jpegs-out-of-thin-air.html

Let's Playwright with .NET 6 MVC

Let's start this story with our protagonist, a consultant in the role of a backend developer with a focus on .NET 6 and Azure. Let us call him Mike. Mike likes to deliver quality. He works with automated unit tests and integration tests and ensures those run in the project's CI pipeline. He uses XUnit and NSubstitute, WireMock.Net and FluentAssertions.

Author Kristof Riebbels

Mike faces a new challenge. Due to colleagues leaving the project, he has been asked to deliver a frontend, made in Dotnet 6 MVC. Our consultant has done some research on web development. Mike liked to test his work when he did backend development.

The transition for Mike to a full-stack developer creates a significant shift in perspective. Mike was only focused on server-side logic and data management. Now he needs to explore and grasp the knowledge of web development and more so discover the common pitfalls.

The goal of this article is to tell the story of Mike's transition from a backend developer to a full-stack developer. The world seemed simpler for him when testing was just an easy thing to do when you code SOLID in a backend environment. UI testing brings its difficulties to the table. Mike will create a Playwright project and discover how Playwright addresses these difficulties.

Visit the dotnet6mvc-Playwright (see References) repository to play with Playwright.


Xpirit

The Backend Developer's and Frontend Developer's Perspective

Mike discovers and feels the extra problems that frontend development brings. He is not a frontend developer, but he is willing to learn the world of frontend development. Let us examine what he has learned so far.

In Mike's world, backend developers are creating applications and by using unit tests they ensure those applications are reliable and resilient. Traditionally, integration tests required a lot of setup and a live environment. However, Mike runs integration tests in his CI pipeline by mocking and working with other services! Mocking HTTP calls is effortless these days. Mike mocks code that integrates with non-HTTP services and he uses mocking frameworks for that. Mike uses in-memory databases so he does not depend on a real database. If a live environment still requires QA testing by potential end users, Infrastructure as Code (IaC) comes to the rescue.

Full-stack developers need to focus on handling the client-side code. Some work with VanillaJS, while others work within a framework that needs to be kept up-to-date. They need to investigate what impact that can have on the end-user. There are many different frameworks to choose from and each comes with ups and downs.

When Mike thinks about frontend development, he notices that the development is predominantly done in JavaScript, a language that offers robust support for asynchronous programming. Developers need to think about the order of execution of the code, the size of the payloads they send to the server and retrieve and limit the number of requests. The UI needs to load as fast as possible, so minifying and splitting up scripts is important.

Some frameworks will generate HTML for you. Mike likes type safety and some tools that help him out. He is only interested in ensuring the REST API is protected by a Bearer token and figuring out what kind of authentication he would use to query databases. Now, Mike needs to think about how to identify users using authentication and authorization flows using the OpenID protocol.

Because the frontend is the first thing that users and hackers are confronted with, frontend developers need to ensure their scripting techniques are up to date and the libraries they use are not vulnerable.

Client-side code needs to run in all kinds of browsers. Is it overkill for Mike to use a cloud tool that offers several different browsers to manually test the application on all kinds of devices?

A little bit of history

Mike wants to understand what happened in the past. He wants to know how the challenges in the past were addressed and where we are today. He believes that understanding the past will help him to understand the present and the future.

In the 1980s and 1990s, as personal computers became more popular, software applications started to have more complex GUIs. This led to the development of automated UI testing tools. These tools allowed testers to record and replay user interactions, making it easier to test complex interfaces.

In the 2000s, when web applications came to be, UI testing evolved again. Web applications have more complex interfaces and are more dynamic than traditional desktop applications. This led to the development of more advanced UI testing tools, like Selenium, that can interact with web elements.

In recent years, with the rise of mobile applications, UI testing has had to adapt yet again. Mobile applications have different interfaces and interaction patterns than web or desktop applications. This has led to the development of new UI testing tools that are designed specifically for mobile applications.

With the rise of working with packages, a frontend developer is capable of reusing components. This means unit testing can be used to test individual components: individual buttons, forms, or other UI elements.

By adding tests for those components, integration tests can be used to test the interaction between components.

Common Problems in UI Testing

Mike went to a conference with some colleagues and attended a talk about UI testing. After that session, he listened closely to what others had to say about the difficulties in UI testing. Below you find a list of what he heard.

- Dynamic content, complex interfaces, cross-browser compatibility, resources, timing, interaction and mobile compatibility are some of the common challenges in UI testing.

- Modern web applications often have dynamic content that changes in response to user interactions. This can make it difficult to write tests that are reliable and repeatable. Dynamic content refers to web pages displaying different text, images, or layouts depending on the user, time of day, or the user's device. Inconsistent identifiers such as IDs, names, and classes might not always be unique or consistent across different versions of the web application, making it challenging for the automation script to locate elements accurately.

- To execute tests, there needs to be a system in place to test with. This system needs to be up to date and running in the correct environment. This can be a challenge in itself. This is where the Backend-For-Frontend (BFF) pattern can help him out. The BFF pattern is a software design pattern that allows developers to create a single backend for one frontend application. This pattern is useful when you have one or multiple frontend applications that need to access the same data or functionality. It can also be used to create a single API for multiple versions of the same frontend application. This way, the backend can easily be mocked out.

- Not all interactions can be tested. If the application communicates with third-party APIs that do not have corresponding data or are not providing a testing environment, it is difficult to test the application.

- Some applications have complex interfaces with many elements. This can make it difficult to write tests that cover all possible user interactions. Some web applications involve complex user interactions such as drag-and-drop, hover menus, or keyboard shortcuts, which can be challenging to automate.

- Different browsers can render web pages in slightly different ways. This can make it difficult to write tests that work correctly on all browsers.

- Mobile devices have different screen sizes and interaction patterns than desktop devices. This can make it difficult to write tests that work correctly on both mobile and desktop devices.

- In Product Development, the continuous evolution and adaptation of products require regular updates and maintenance of UI tests, but the fragility and high maintenance requirements can decrease motivation to develop them. In Project Development, the predefined scope and limited changes can lead to minimal redundancy in UI tests, but strict contracts or tight deadlines may hinder the motivation to write them due to difficulties in modifying tests later on...

Playwright is the new kid on the block

As Mike deepens his understanding of frontend development, he realizes the benefit of tools like Selenium and Playwright for component testing and end-to-end user interaction simulations. He discovers Selenium to be a well-established framework. Selenium has a reputation for being reliable and versatile. Selenium offers cross-browser testing and supports a large range of programming languages. It facilitates frontend UI testing across actual servers and cloud-based, real-device testing.

Despite Selenium's reputation, Mike finds himself leaning towards Playwright. Developed by Microsoft, Playwright offers certain advantages that appeal to him. Playwright's support for headless browser architecture allows for a quicker feedback cycle. This is a useful (quality of life) feature for a backend developer learning frontend development! Playwright's automatic waiting mechanism reduces the instability in tests. He read about the isolated browser contexts. These isolated contexts let you conduct tests independently without any shared state and simultaneous user logins. Debugging becomes simpler as well. Mike would not need to worry about the residual effects from previous tests. Playwright can emulate different devices and geolocations. These features allow him to recreate all kinds of user scenarios.

Enter Playwright

Playwright is an open-source Node library developed by Microsoft that allows developers to automate web browsers over the Chromium, Firefox, and WebKit protocols. It provides capabilities to interact with web pages, evaluate scripts, generate screenshots, and produce PDFs. It's used for end-to-end testing of web applications to ensure their correct functionality across different web browsers.

Playwright evolved from the Puppeteer project. That was limited to Chrome automation. Microsoft's effort with Playwright aim to address the multi-browser scenario, making it possible to run the same tests on different browsers without any code changes. This is a leap forward as many businesses need to ensure their web applications work seamlessly across all major browsers.

The .NET community showed interest in having Playwright's capabilities within their ecosystem. Microsoft recognized this demand and introduced Playwright for .NET, allowing .NET developers to write tests in C#.

Playwright for .NET is a client package that allows communication with the Playwright Node.js Server. Instead of writing Mike's tests in JavaScript, it brings the Playwright API to .NET developers. Because it is a client-server model, Mike has the same underlying browser automation engine and thus he can use the same capabilities.

Features that make Playwright great

Mike wants to know what he can do with Playwright and discovers the following features on the Playwright website:

- Playwright automates the Chromium, WebKit, and Firefox browsers with a single API to cover all rendering engines.
- Playwright allows testing of how an application behaves on different devices by adjusting the viewport size of the browser.
- Playwright also allows for network throttling, where developers can simulate slow network connections and assess the impact on the application's performance. Using Playwright's built-in network management features, developers can emulate slow or offline network conditions to measure how the application endures under different scenarios.
- To speed up UI testing, developers can employ parallel test execution. Playwright's auto-wait mechanism and support for intercepting network requests make it ideal for testing single-page applications (SPAs). Developers can ensure critical page elements are available and the application is making the expected API calls during navigation and user interactions.
- Playwright enables developers to automate testing form submission and validation.
- Playwright offers to reuse the authentication of the browser, making it easier to test applications.
- Playwright allows you to automate browser interactions, and you can run those headed or headlessly.
- Playwright has some features for capturing screenshots and recording videos of your browser sessions. This will help out when it comes to debugging, documentation, or even visual verification. Combine this strategy with a CI/CD pipeline, and you have more context when a test fails.

Headed vs Headless

Alright, Mike dives into the terms `headless` and `head` in the context of browsers and unit testing, and Playwright.

When Mike runs a browser in `headed` mode, it means Mike is getting the entire graphical user interface. Mike sees the web pages loading and he can click around — the whole shebang.

In `headless` mode, the browser runs without a GUI. It's all happening in the background, so Mike can't see it, but it's there doing its thing. This is super useful for automated tasks, server environments, or testing scenarios where you don't need the GUI.

In unit testing, `headless` and `headed` usually tie back to how tests run on a browser. When Mike's tests run in a visible browser window (`headed`), he can watch as the browser navigates through the test steps. It's slower but good for debugging.

When the browser remains in the background (`headless`), Mike does not see any GUI. Tests run faster this way, which is ideal for CI/CD pipelines where Mike just wants to know if things pass or fail without the visual overhead.

Mike's introduction to UI Testing with Playwright in .NET 6 MVC

Let's follow Mike's steps on how to get started with Playwright in .NET 6 MVC using Playwright's documentation.

Mike searched and reused somebody's web application. He found an e-commerce website written in Dotnet 6 MVC. It is a small application where the user needs to be created and be authorized to view, create and/or update a list of products. Mike ensures the application is running and he can access the webpage.

Mike reads that Playwright for .NET works best with NUnit. While Playwright supports other test runners like MSTest, Mike will use NUnit. The Playwright's test runners' key focus is to optimize test performance by reusing Playwright and Browser instances and running each test case in a new `BrowserContext` to isolate browser states.

Playwright does not support the parallelization of tests. By default, NUnit, MSTest and XUnit will run all test files in parallel. Playwright offers support for configuring NUnit and MSTest so each test within a test file is running sequentially. To set up NUnit, there is an option `ParallelScope.Self` to create as many processes as there are cores on the host system. Running tests in parallel using `ParallelScope.All` or `ParallelScope.Fixtures` are not supported.

Mike followed Playwright's tutorial with ease. He copies a test that visits[1] and validates the title of the homepage. When he ran the test, he was happy it turned green.

However, he started to question the tool. He did not see anything happen. He knew that the tool had support for creating screenshots. He added a line of code that will take a print screen from the page. The screenshot will be saved in the `bin/Debug/net6.0` folder.

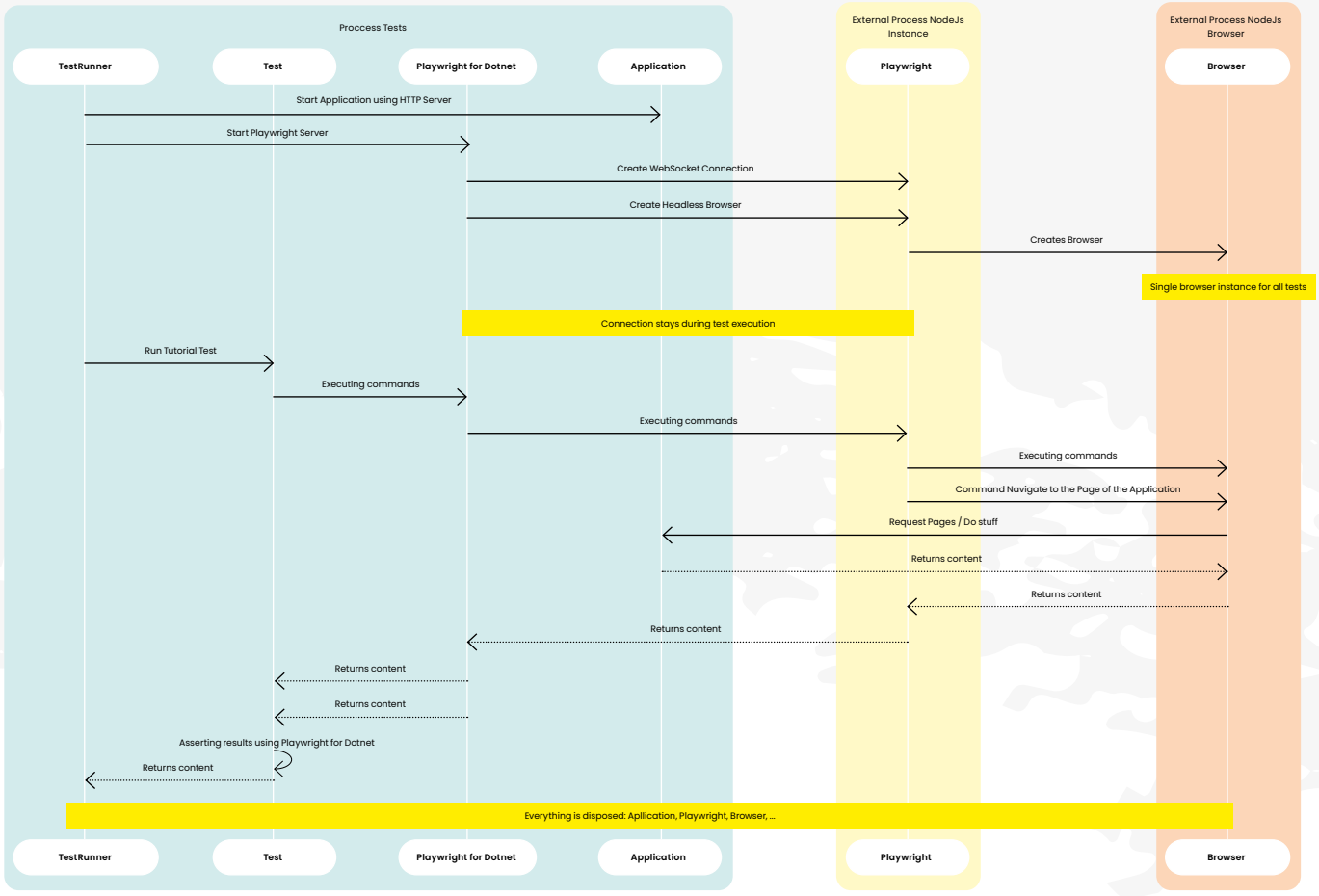
```
await Page.ScreenshotAsync(new PageScreenshotOptions {
Path = "image.png" });
```

He ran the test again and saw the screenshot appear in the `bin/Debug/net6.0` folder

Mike browsed the debug folder and he noticed Playwright-related files and folders:

- a folder called `.playwright`: This folder contains two other folders that contain NodeJS and the Playwright code
- a file named `playwright.ps1`: A PowerShell file that will execute the method `Program.Main` in the `Microsoft.Playwright.dll`.
- Microsoft Playwright DLLs: The code that ensures communication with the Playwright NodeJs Server.

Because Mike discovered the `.playwright` folder, he became curious about how the code in the test behaves. To understand this process, he needs to understand the architecture. Mike discovered that Playwright communicates all requests through a single web socket connection. That connection stays in place until the test execution is completed. This reduces points of failure and allows commands to be sent quickly over a single connection. Playwright also uses a single browser instance for all tests. That reduces the overhead of creating and destroying browser instances.



[1] https://Playwright.dev

Mike is now ready to write his tests. He wants to test the application's login page. To achieve that goal, he needs to start the e-commerce application. He read the Microsoft documentation on how to create integration tests and started the application using the `WebApplicationFactory`.

The `WebApplicationFactory` serves as an in-memory host for Mike's web application. What sets the `WebApplicationFactory` apart is its usage of a `DeferredHostBuilder`. The web application is started right before the `HttpClient` is created. The sequences of method calls occurring in `Program.cs` are recorded by the `WebApplicationFactory`, without executing them. This grants Mike the flexibility to override registered services, which is useful to ensure the application and tests do not access third parties (e.g. a database).

Mike creates a `HttpClient` using the `WebApplicationFactory.CreateClient` to access the e-commerce webpage and he retrieves the homepage! When he started to write his first Playwright tests, he noticed the following error message.

```
Message: Microsoft.Playwright.PlaywrightException :
net::ERR_CONNECTION_REFUSED at http://localhost/
========================== logs ==========================
navigating to "http://localhost/", waiting until "load"
==========================================================
```

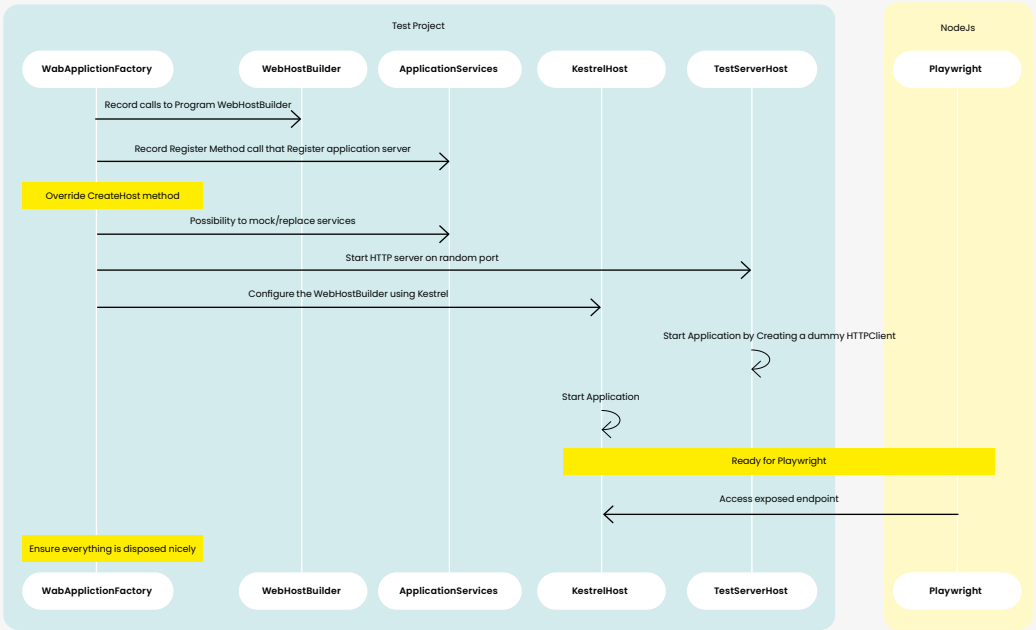
Searching the internet, Mike discovered that the `WebApplicationFactory` is not a great fit **at this moment**. That class is tightly coupled with the HTTP Server called `TestServer`. The `TestServer` can host our application and is approachable from the `HttpClient` that is created by the method `WebApplicationFactory.CreateDefaultClient`.

External processes, such as Playwright, cannot access the e-commerce web application.

One solution is using `Kestrel`. `Kestrel` can expose the application's endpoints and pages. Playwright can then interact with the application. When Mike investigated further, he read some threads on the issues list of dotnet on GitHub. Microsoft will do some refactoring, however, this is not a priority for DotNet 7 and seems it is in triage for DotNet 8 **at this moment**.

Mike creates a class that extends the `WebApplicationFactory` called `PlaywrightCompatibleWebApplicationFactory`. When you look at the code below, Mike noticed the creation of two Hosts.

```
protected override IHost CreateHost(IHostBuilder
builder) {
    try {
        _hostThatRunsTestServer = builder.Build();
        builder.ConfigureWebHost(webHostBuilder =>
        webHostBuilder.UseKestrel());
        _hostThatRunsKestrelImpl = builder.Build();
        _hostThatRunsKestrelImpl.Start();
        var server = _hostThatRunsKestrelImpl.Services.
        GetRequiredService();
        var addresses = server.Features.Get();
        ClientOptions.BaseAddress = addresses!.
        Addresses.Select(x => new Uri(x)).Last();
        _hostThatRunsTestServer.Start();
        return _hostThatRunsTestServer;
    }
    catch (Exception e) {
        _hostThatRunsKestrelImpl?.Dispose();
        _hostThatRunsTestServer?.Dispose();
        throw;
    }
}
```

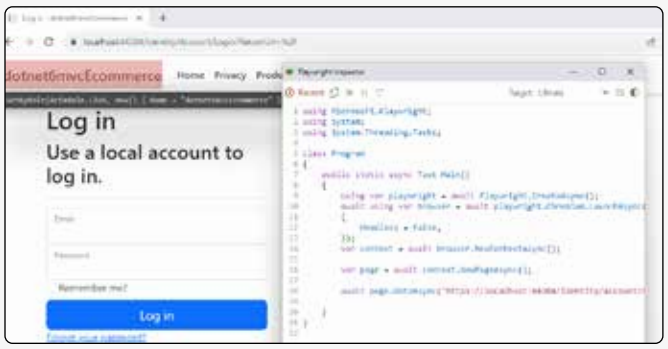


The application wants to create a `Host` that encapsulates the HTTP server `TestServer`. This means Mike needs to create an extra HTTP server `Kestrel`. Mike needs that because `WebApplicationFactory` exposes a property called `Server`. That property exposes the type `TestServer` and not the type `IServer` The `estServer`'s Host is created first.

If Mike configured `Kestrel` on the builder first, then he retrieves an instance of `KestrelImpl`, he cannot return an instance of the type `TestServer`.

Now the application is accessible to the outside world and thus for Playwright, he wanted to write some tests. He started with a simple test that will visit the login page and validate the title of the page. To achieve this, he uses the code generator that Playwright offers. By executing `.\Playwright.ps1 codegen`, a window called `Playwright Inspector` appeared.

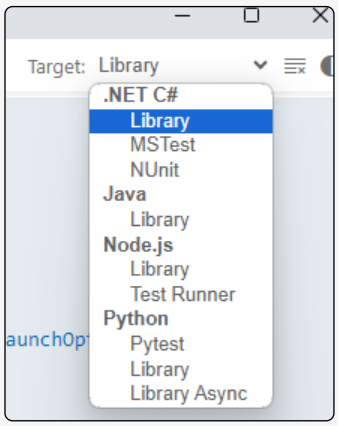
In that window, Mike noticed some code it generates for you:



Code generation on startup

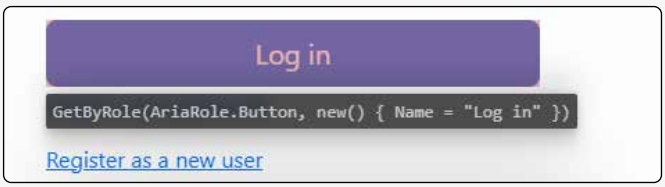
When Mike clicked on the dropdown box Target, he noticed a list of languages: he can use C#, Java, Python or JavaScript.

Mike is not familiar with those other languages, so he will stick with C# and choose NUnit.



Supported languages

Mike hovered over the elements on the browser and noticed that Playwright marked the element. Playwrights added a label below the marked element. That label contains the locator to fetch that element. Mike can copy that locator and use it in his tests.



Hovering with the mice over an element and getting the locator

Mike clicked and pressed some keys while recording his actions to achieve his first test: **Not able to log in with the wrong credentials.**

```
[Test]
public async Task MyTest() {
    await Page.GotoAsync("https://localhost:44304/Identity/Account/Login?ReturnUrl=%2F");
    await Page.GetByLabel("Email").ClickAsync();
    await Page.GetByLabel("Email").FillAsync("test@test.be");
    await Page.GetByLabel("Email").PressAsync("Tab");
    await Page.GetByLabel("Password").FillAsync("ABc.123!");
    await Page.GetByRole(AriaRole.Button, new() { Name = "Log in" }).ClickAsync();
    await Page.GetByText("Invalid login attempt.").ClickAsync();
}
```

With some adjustments, Mike manually created two tests from the recorded user interactions using the `Playwright Inspector`.

• One to verify if the redirection is working

```
[Test]
public async Task WhenProvidingBaseUrl_ShouldRedirectToLoginPage() {
    await Page.GotoAsync(_webApplicationFactory.ServerAddress); //Should be redirected.
    await Expect(Page).ToHaveURLAsync(_webApplication-Factory.ServerAddress+"Identity/Account/Login?
    ReturnUrl=%2F");
  }
```

• One to verify if the login is working when the wrong credentials are provided

```
[Test]
public async Task WhenProvidingWrongCredentials_ShouldRespondWithInvalidLoginAttempt() {
    await Page.GotoAsync(_webApplicationFactory.ServerAddress);
    await Page.GetByLabel("Email").ClickAsync();
    await Page.GetByLabel("Email").FillAsync("test@test.be");
    await Page.GetByLabel("Email").PressAsync("Tab");
    await Page.GetByLabel("Password").FillAsync("ABc.123!");
    await Page.GetByRole(AriaRole.Button, new() { Name = "Log in" }).ClickAsync();
    await Expect(Page.GetByText("Invalid login attempt.")).ToBeVisibleAsync();
  }
```

When Mike ran the test, he saw that everything just worked. He wanted to view the actions in the test coming to life before his eyes. Mike followed the documentation and decided to use the `.runsettings` file. He configures Visual Studio by clicking on Test > Configure Run Settings > Select Solution Wide Settings > Select the `runsettings` file.

By applying the default settings, the browser appeared and he saw the test executing the actions. This is because `Playwright.LaunchOptions.Headless` is set to `false`. The `DEBUG` environment variable is set to `pw:api` to get more information about the API calls that are made.

```
<?xml version="1.0" encoding="utf-8"?>
<RunSettings>
  <!-- NUnit adapter -->
  <NUnit>
    <NumberOfTestWorkers>24</NumberOfTestWorkers>
  </NUnit>
  <!-- General run configuration -->
  <RunConfiguration>
    <EnvironmentVariables>
      <!-- For debugging selectors, it's recommended to set the following environment variable -->
      <DEBUG>pw:api</DEBUG>
    </EnvironmentVariables>

  </RunConfiguration>
  <!-- Playwright -->
  <Playwright>
    <BrowserName>chromium</BrowserName>
    <ExpectTimeout>5000</ExpectTimeout>
    <LaunchOptions>
      <Headless>false</Headless>
      <Channel>msedge</Channel>
    </LaunchOptions>
  </Playwright>
</RunSettings>
```

Mike added the PWDEBUG environment variable with the value `console`. That allowed him to debug the selectors in the console of the browser using the variable `playwright`.

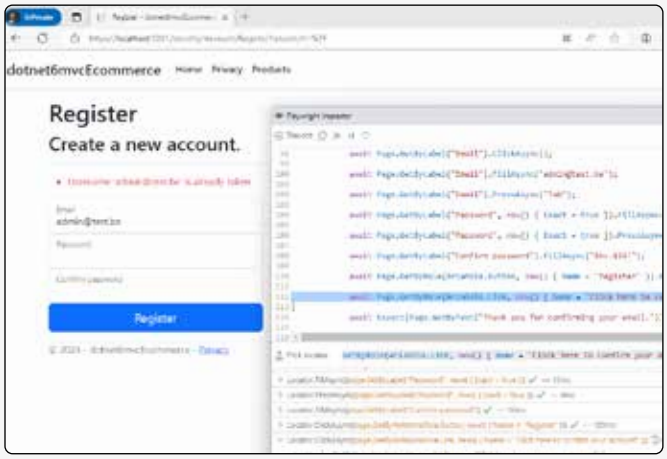


Figure 4: Alt text

Mike has added the environment variable PWDEBUG with value 1, ran the test and the `Playwright Inspector` opened up. He stepped through using the popular `F10` key. Mike saw the `Playwright Inspector` in action. In this test case, he noticed a problem. The Username `admin@test.be` is already used in a registration.

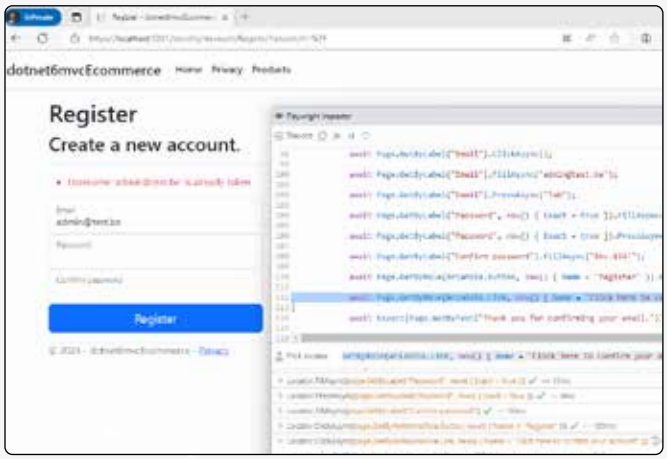


Figure 5: Alt text

For this test, no further help was needed, but he was curious about another tool called Trace Viewer. That tool should help in diagnosing and fixing problems. When recording a trace, it captures a snapshot of the page after every action and records network requests, JavaScript logs, etc. Mike browses the `BrowserContext` using IntelliSense:

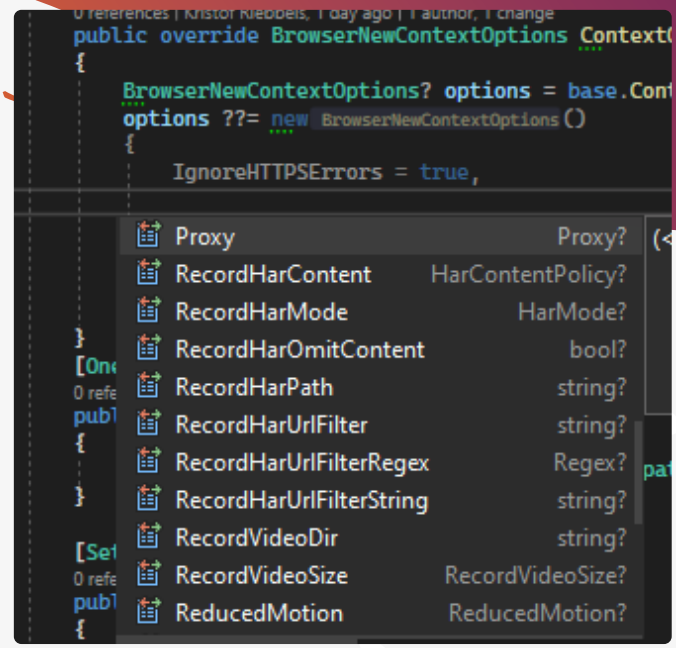


Figure 6: Alt text

Mike did want to know more about Playwright and how it could help him with automating the authenticating of a user so he could test his creating/editing and listing products. He found two methods that can help him with that.
To authenticate, Mike can fill and submit login forms as he did before:

```
await Page.GotoAsync(_webApplicationFactory.
ServerAddress);
await Page.GetByLabel("Email").ClickAsync();
await Page.GetByLabel("Email").FillAsync("test@test.be");
await Page.GetByLabel("Email").PressAsync("Tab");
await Page.GetByLabel("Password").FillAsync("ABc.123!");
await Page.GetByRole(AriaRole.Button, new() {
Name = "Log in" }).ClickAsync();
```

Another method is to restore cookies and local storage. Because a test should only do what it states, he liked this functionality. If there is already a test that covers the login functionality, then there is no need to test the same functionality again in another test.

After a successful login, Mike saved the state from the cookies and local storage and reused it instead of logging in each time. The method `BrowserContext.StorageStateAsync` is helpful for that.

Playwright also mentions that Mike can manipulate the `sessionStorage` of your browser. The method `Page.EvaluateAsync` helps you with that.

```
string sessionStorageData = await Page.Evaluate-
Async<string>("() => JSON.stringify(window.session-
Storage)");
```

In Playwright's documentation, Mike found code that executes JavaScript when the page is being initialized. It will set the `sessionStorage` when the page is loading.

Mike still had one more splinter in his brain. He found confidence in writing and debugging tests but what about running it in a CI Pipeline? Playwright has a lot of samples on how to use a CI Pipeline on Azure, GitHub or other CI tools. Mike uses the GitHub Actions sample. However, an error occurred:

`The argument 'bin/Debug/net6.0/playwright.ps1 is not recognized as the name of a script file.`

A quick search on the internet and Mike found a solution. He needed to add the following line to the test project file:

```
<PlaywrightPlatform>all</PlaywrightPlatform>
```

He added the path to the test project `dotnet6mvc-Ecommerce.Playwright.tests/bin/Debug/net6.0/playwright.ps1` as well as updated Powershell.

```
  .- run: dotnet tool update --global PowerShell
```

Mike ran the GitHub action again and it worked!

```
name: Ecommerce Playwright Tests
on:
  push:
    branches: [ main, master ]
  pull_request:
    branches: [ main, master ]
jobs:
  test:
    timeout-minutes: 60
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Setup dotnet
      uses: actions/setup-dotnet@v3
      with:
        dotnet-version: 6.0.x
    - run: dotnet tool update --global PowerShell
    - run: dotnet build
    - name: Ensure browsers are installed
      run: pwsh dotnet6mvcEcommerce.Playwright.
      tests/bin/Debug/net6.0/playwright.ps1 install
      --with-deps
    - name: Run your tests
      run: dotnet test
```

Conclusion

Mike is happy that he has a way to quickly generate code from his interactions. He will use the recording, tracing and debugging features so he can start writing his tests. Playwright is a great tool and it is easy to use. Every example and tutorial Mike found on the website just works outside the box! I hope that you, like Mike, are inspired to give Playwright a try. When you have feedback, do not hesitate to contact me. We all learn from each other. </>

References

- https://news.ycombinator.com/item?id=27460329
- https://github.com/microsoft/Playwright
- https://learn.microsoft.com/en-us/aspnet/core/test/integration-tests?view=aspnetcore-7.0
- https://blog.martincostello.com/integration-testing-antiforgery-with-application-parts/
- https://danieldonbavand.com/2022/06/13/using-Playwright-with-the-webapplicationfactory-to-test-a-blazor-application/
- https://github.com/dotnet/aspnetcore/issues/33846
- https://www.meziantou.net/automated-ui-tests-an-asp-net-core-application-with-Playwright-and-xunit.htm
- https://medium.com/younited-tech-blog/end-to-end-test-a-blazor-app-with-Playwright-part-1-224e8894c0f3
- https://learn.microsoft.com/en-us/visualstudio/test/configure-unit-tests-by-using-a-dot-runsettings-file?view=vs-2022
- https://github.com/kriebb/dotnet6mvc-Playwright
- https://aws.amazon.com/blogs/mobile/backends-for-frontends-pattern
- https://research.aimultiple.com/playwright-vs-selenium/
- https://www.browserstack.com/guide/playwright-vs-selenium
- https://www.linkedin.com/pulse/selenium-master-automation-qualitymatrix/

Sustainable Software Engineering Through the Lens of Environmental

Did you know that sustainable software engineering is a topic we frequently discuss and engage with? However, our conversations predominantly revolve around the economic dimension, such as optimizing costs in cloud computing, or the technical dimension, particularly when addressing code maintainability. But were you aware that sustainable software engineering encompasses five distinct dimensions?

Author Danny van der Kraan


Kevin
ESTD·2014
Kevin Originals

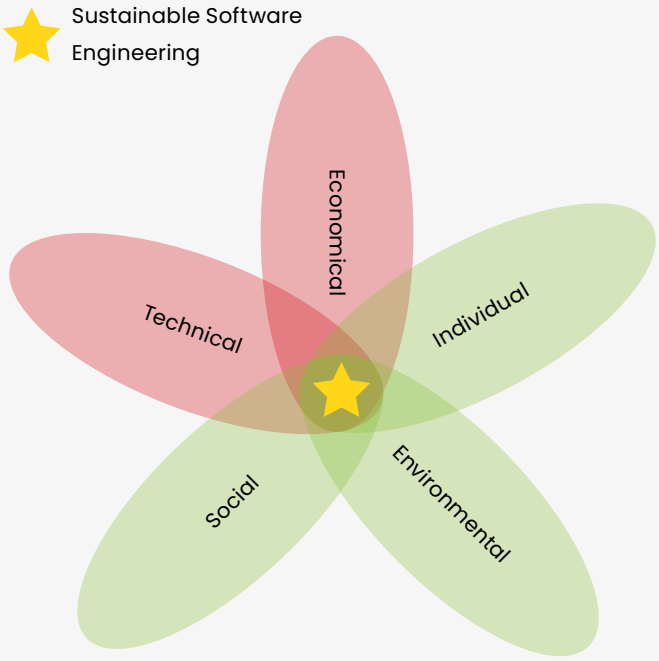


Figure 1: Dimensions Sustainable Software Engineering
Source: https://se.ewi.tudelft.nl/research-lines/sustainable-se/

The remaining three dimensions are individual, social, and environmental. In this article, we will shift our focus to the latter dimension and explore how we can nurture a greener environment through software engineering, paving the way for 'GreenOps' (yes, another 'Ops' term!). The best part? It's not as daunting as it may seem!

Sustainable Software Engineering Environmental Dimension?

When addressing sustainable software engineering within the environmental context, we are essentially examining the software's impact on the environment. This impact can be substantial. For instance, were you aware that operational software contributes to a significant 2-3% of global CO_2 emissions[1]? To put it into perspective, this level of emissions is on par with that of the aviation industry[2]. Consequently, when we delve into the realm of sustainable software engineering with environmental concerns in focus, our primary objective revolves around reducing CO_2 emissions. We can achieve this, for example, by optimizing hardware utilization to minimize e-waste or by enhancing the energy efficiency of our software. These are the key areas we will explore in this article. The good news is that we don't have to start entirely from scratch.

[1] https://techmonitor.ai/focus/tech-industry-carbon-emissions-progress
[2] https://www.iea.org/energy-system/transport/aviation

The Green Software Foundation and Microsoft's Well Architected Framework

The Green Software Foundation (GSF) is a non-profit organization dedicated to mitigating the environmental impact of software[3] (https://greensoftware.foundation/). They achieve this mission by furnishing a framework for green software engineering.

Figure 2: The Green Software Foundation

This framework rests upon six core principles:

- **Carbon Efficiency**: Strive to emit the least possible amount of carbon.
- **Energy Efficiency**: Endeavor to use the minimum amount of energy necessary.
- **Carbon Awareness**: Adjust operations based on electricity cleanliness; do more when it's cleaner and less when it's dirtier.
- **Hardware Efficiency**: Minimize the embodied carbon in hardware usage.
- **Measurement**: Understand that what you cannot measure, you cannot improve.
- **Climate Commitment**: Gain a deep understanding of the precise mechanisms behind carbon reduction.

Additionally, GSF has meticulously documented cloud-agnostic patterns aligned with each principle.
These principles and patterns have been integrated into practices that can be readily applied within Microsoft's Azure cloud infrastructure, following the Well Architected Framework. This comprehensive framework outlines best practices for the development of cloud-native applications and includes an entire section dedicated to sustainable workloads. Moving forward in this article, we will delve into these principles, patterns, and practices on Azure.
Without further delay, let's commence with an exploration of the first principle, an accompanying pattern, and a tangible implementation on the Azure platform.

Carbon Efficiency

Carbon efficiency pertains to the efficiency of a process, product, or organization in minimizing carbon emissions while still achieving its objectives. This constitutes a fundamental principle within environmental sustainability initiatives and endeavors to combat climate change.
The central concept revolves around the minimization of the carbon footprint linked to any unit of work to the greatest extent possible.

Pattern: Data Lifecycle Management

One effective pattern to strive for in order to achieve carbon efficiency is data lifecycle management. Data lifecycle management encompasses the comprehensive process of overseeing data from its inception to its deletion.
This entails careful consideration of how data is stored, processed, and analyzed. The primary objective of data lifecycle management is to ensure the efficient and effective utilization of data while concurrently minimizing its environmental footprint, particularly within the context of this article. This emphasis on efficient data management stems from the realization that both the processing and storage of data consume energy, consequently contributing to carbon emissions.

Practice: Azure Storage Lifecycle Management

Azure Storage lifecycle management provides a rule-based policy that enables you to manage blob data by transitioning it to the appropriate access tiers or expiring data when it reaches the end of its lifecycle. Using this lifecycle management policy, you can:

- Swiftly transition blobs from cool or cold or archive storage tiers to hot storage when they are accessed, optimizing for performance.
- Move current versions of a blob, previous versions, or blob snapshots to a cooler storage tier if these objects have not been accessed or modified for a specified period, thereby optimizing for cost.
- Automatically delete current versions of a blob, previous versions, or blob snapshots when they reach the end of their respective lifecycles.
- Establish the aformentioned rules to be executed daily at the storage account level.
- Apply rules selectively to containers or a subset of blobs using criteria such as name prefixes or blob index tags.

[3] https://greensoftware.foundation/

Lifecycle management policies can be automated.
Here's an example of how to achieve this in Terraform.

```
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.0"
    }
  }
}

provider "azurerm" {

  features {}
}

resource "random_id" "id" {
  byte_length = 8
}

resource "azurerm_resource_group" "sustainability-example" {
  name     = "rg-${random_id.id.hex}"
  location = "West Europe"
}

resource "azurerm_storage_account" "sustainability-example" {
  name                = "sa${random_id.id.hex}"
  resource_group_name = azurerm_resource_group.sustainability-example.name

  location                 = azurerm_resource_group.sustainability-example.location
  account_tier             = "Standard"
  account_replication_type = "LRS"
  account_kind             = "BlobStorage"
}

resource "azurerm_storage_container" "sustainability-example" {
  name                  = "examplecontainer"
  storage_account_name  = azurerm_storage_account.sustainability-example.name
  container_access_type = "private"
}

resource "azurerm_storage_management_policy" "sustainability-example" {

  storage_account_id = azurerm_storage_account.sustainability-example.id

  rule {
    name    = "MoveToCoolStorage"
    enabled = true
    filters {
      prefix_match = [azurerm_storage_account.sustainability-example.name]
      blob_types   = ["blockBlob"]
    }
    actions {
      base_blob {
        tier_to_cool_after_days_since_modification_greater_than = 2
      }
      snapshot {
        delete_after_days_since_creation_greater_than = 2
      }
    }
  }
}
```

This example provides all the essential components to kickstart your project. It initiates the process by generating a Resource Group suffixed with a random number, utilizing the Terraform resource `random_id`. Subsequently, it establishes a Storage Account with the account kind set to `BlobStorage`. Within this Storage Account, a container is created.

The pivotal segment of this setup lies at the end, where we leverage the Terraform resource `azurerm_storage_management_policy` to define a lifecycle management policy for the previously created Storage Account. This policy incorporates a rule that triggers an action to transfer the blob to the 'cool' access tier if it remains unmodified for a duration of 2 days, aptly named `MoveToCoolStorage`. Additionally, it implements an action to remove snapshots after a 2-day period.

This comprehensive configuration will set the foundation for efficient data management within your Azure environment.

Energy Efficiency

The principle of energy efficiency is founded on the concept that we can curtail our energy consumption by employing the minimal amount of energy required to attain equivalent or superior outcomes. Consequently, this approach aids in diminishing our carbon footprint, as energy consumption serves as a reliable proxy for carbon emissions.

Pattern: Energy Proportional Computing

Energy proportional computing is a pattern that aims to reduce energy consumption in computing systems by ensuring that the energy consumption of the system is proportional to its workload.

Practice: Azure Container Apps Jobs

Achieving energy proportional computing is possible by leveraging Azure Container Apps Jobs, which allow you to execute containerized tasks for a defined duration before terminating. These jobs can be employed for various tasks, including data processing, machine learning, or any scenario requiring on-demand processing. Each job execution typically handles a single unit of work.

Figure 3: Azure Container Apps Jobs

Job executions can commence manually, follow a predefined schedule, or trigger in response to specific events. These jobs encompass various tasks, such as on-demand batch processes and scheduled activities. Essentially, when the system's workload is light, it should consume minimal energy, and when the workload i ntensifies, its energy consumption should proportionally increase. This scalability in energy consumption, aligned with workload fluctuations, exemplifies an energy-efficient system.

Carbon Awareness

The principle of carbon awareness revolves around the concept that we can diminish our carbon emissions and contribute to combating climate change by aligning our energy consumption with the availability of clean energy sources. In essence, this entails utilizing more energy during periods when renewable sources, such as wind and solar, are producing the highest electricity output, while conserving energy during times when fossil fuels are the primary energy generation source.

Pattern: Process when the carbon intensity is low

Carbon intensity is a measure of the amount of carbon dioxide (CO_2) emitted per unit of energy produced. This pattern involves scheduling computing workloads to run during times when the carbon intensity of the energy grid is low. The carbon intensity of the energy grid varies depending on factors such as the time of day, weather conditions, and energy demand.

Practice: Use Carbon Aware SDK CLI in your pipeline to deploy to regions with low carbon intensity

The Carbon Aware SDK CLI is a valuable tool designed to assist developers in deploying their applications to regions characterized by low carbon intensity. Its functionality hinges on the analysis of carbon intensity across various regions, enabling the deployment of applications to the region exhibiting the least carbon intensity. This strategic approach substantially contributes to minimizing the overall carbon footprint associated with the application.

However, it's essential to acknowledge that this solution may not be suitable for every workload. Organizational policies may exist that prohibit deployments in certain regions, thereby limiting its applicability. As an illustrative scenario, consider a situation where you're deploying Template Specs and conducting automated testing within your Azure pipeline. In such cases, the Carbon Aware SDK can be employed to ensure that test Template Specs are deployed in regions characterized by low carbon intensity, aligning with sustainability objectives.

Hardware Efficiency

The principle of hardware efficiency is centered around the reduction of embodied carbon during both the production and utilization of hardware. Consumption related to cloud-based services, encompassing servers, network cables, and other components, contributes significantly to embodied carbon. Sustainable software engineering in cloud environments can play a pivotal role in assisting businesses in curbing their carbon footprint by optimizing hardware usage to minimize embodied carbon to the greatest extent possible.

Pattern: Use Spot Instances When Possible

Spot instances represent a feature available from cloud computing providers like Amazon Web Services (AWS) and Google Cloud Platform (GCP), enabling users to leverage surplus computing capacity. Typically, these instances are referred to as "spot" instances due to their availability and pricing that can vary based on demand. The outcome is substantial cost savings, particularly for workloads that don't have stringent time constraints and can withstand

interruptions. This approach also enhances the utilization of hardware resources, leading to a reduction in embodied carbon, which aligns well with the themes explored in this article.

Practice: Spot Containers on Azure

Azure Container Instances (ACI) Spot Containers offer the capability to execute interruptible workloads in a containerized format, leveraging unused Azure capacity. They combine the simplicity of ACI with the cost-effectiveness associated with Spot VMs, which is particularly noteworthy as they come at a lower cost compared to regular ACI containers. Azure Container Instances Spot Containers offer support for both Linux and Windows containers, ensuring flexibility across various operating system environments. It's important to note that, unlike Spot VMs, you can't select eviction types or policies for these containers. In the event of an eviction, the container groups hosting the workloads are automatically restarted without any manual intervention. However, it's crucial to acknowledge that spot containers may not be suitable for all types of workloads, as they can be interrupted at any time. Therefore, it's advisable to design your applications to gracefully handle interruptions. As an example, consider a scenario where:

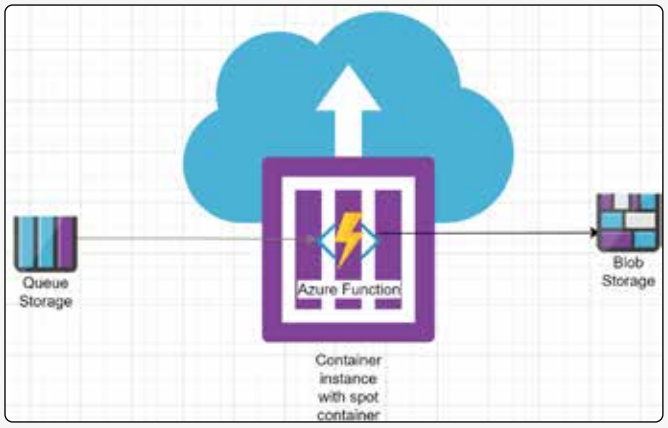


Figure 4: Azure Container Instances Spot Containers

You have a Queue Storage containing messages, and a Function App triggered by the Queue Storage. The Function App performs various processing tasks, such as storing a blob in a Blob Storage, before completing its execution by acknowledging the message in the queue. In the event the spot container is evicted before the Function App can finish processing, the message will not be acknowledged and will remain in the queue. This illustrates how interruptible workloads can be structured.

Measurement

This principle underscores the significance of collecting and analyzing data related to energy consumption, infrastructure utilization, and performance metrics. By doing so, businesses can pinpoint areas where they can optimize both their software and infrastructure to reduce their environmental impact.

Pattern: Carbon Emissions Tracking

By actively tracking carbon emissions, software engineers can identify specific areas where emissions can be curtailed and then take steps to mitigate their environmental footprint. The key components of this process include establishing a baseline, measuring emissions, analyzing the resulting data, and generating reports that inform actionable measures.

Practice: Emissions Impact Dashboard on Azure

The Emissions Impact Dashboard, a tool provided by Microsoft Azure, facilitates the monitoring and analysis of carbon emissions linked to the use of Azure services. This dashboard offers a comprehensive array of metrics and visualizations that aid in comprehending the carbon footprint associated with service usage and identifying opportunities for emission reductions. A section of the dashboard is showcased below:

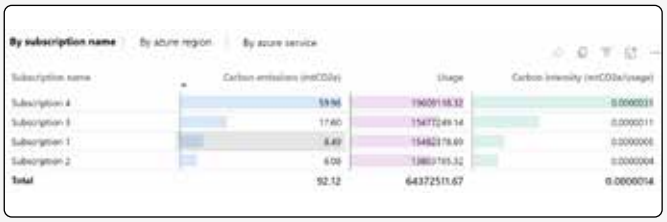


Figure 5: Emissions Impact Dashboard on Azure

Within this section of the dashboard, users can view carbon emissions data across their usage, thereby obtaining insights into the carbon intensity score per subscription. This initial data can serve as a starting point for identifying areas where optimization can be pursued.

Climate Commitment

In the context of sustainable development, climate commitment necessitates that businesses proactively engage in efforts to diminish their carbon footprint. This commitment may encompass the establishment of carbon reduction targets, the adoption of energy-efficient technologies and practices, and investments in renewable energy sources. Pertaining to cloud computing, climate commitment holds significant relevance for sustainable software engineering. Through a comprehensive understanding of the precise mechanisms behind carbon reduction, businesses can effectively determine the most advantageous strategies for minimizing their carbon footprint in relation to their utilization of cloud services.

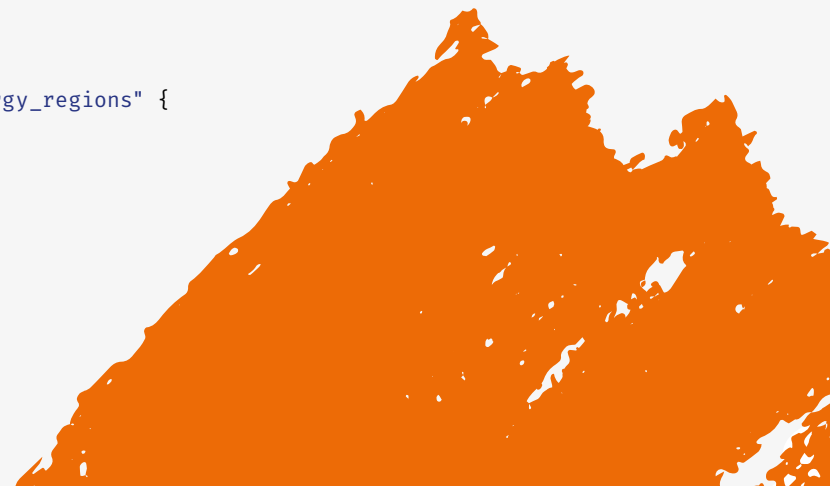
Pattern: Define Policies

Policies constitute a set of rules governing the behavior of a system. They are instrumental in enforcing security, compliance, and other requirements. In the realm of sustainable software engineering, policies can be harnessed to outline the desired behavior of a system with regard to carbon emissions.

Practice: Azure Policy

One practical approach involves establishing an Azure Policy initiative focused on "Sustainability". This initiative can encompass various policies, as illustrated in the example below using Terraform:

```
terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.0"
    }
  }
}

provider "azurerm" {
  features {}
}

resource "azurerm_policy_definition" "energy_efficient_vm_sizes" {
  name         = "energy-efficient-vm-sizes"
  policy_type  = "Custom"
  mode         = "Indexed"
  display_name = "Energy Efficient VM Sizes"

  metadata = jsonencode({
    version  = "1.0.0"
    category = "Sustainability"
  })

  policy_rule = <<POLICY_RULE
{
  "if": {
    "allOf": [
      {
        "field": "type",
        "equals": "Microsoft.Compute/virtualMachines"
      },
      {
        "not": {
          "field": "Microsoft.Compute/virtualMachines/sku.name",
          "in": ["Standard_B1ls", "Standard_B1s", "Standard_B1ms", "Standard_B2s", "Standard_B2ms"]
        }
      }
    ]
  },
  "then": {
    "effect": "deny"
  }
}
POLICY_RULE
}

resource "azurerm_policy_definition" "renewable_energy_regions" {
  name         = "renewable-energy-regions"
  policy_type  = "Custom"
  mode         = "Indexed"
  display_name = "Renewable Energy Regions"

  metadata = jsonencode({
    version  = "1.0.0"
    category = "Sustainability"
  })
```



```
  policy_rule = <<POLICY_RULE
{
  "if": {
    "allOf": [
      {
        "field": "location",
        "notIn": ["westeurope", "uksouth", "northeurope", "eastus", "westus2", "canadacentral"]
      },
      {
        "field": "type",
        "equals": "Microsoft.Compute/virtualMachines"
      }
    ]
  },
  "then": {
    "effect": "deny"
  }
}
POLICY_RULE
}

resource "azurerm_policy_set_definition"
"sustainable_initiative" {
  name         = "sustainable-initiative"
  policy_type  = "Custom"
  display_name = "Sustainable Initiative"

  metadata = jsonencode({
    version  = "1.0.0"
    category = "Sustainability"
  })

  policy_definition_reference {
    policy_definition_id = azurerm_policy_definition.energy_efficient_vm_sizes.id
    reference_id         = "energy-efficient-vm-sizes"
  }

  policy_definition_reference {
    policy_definition_id = azurerm_policy_definition.renewable_energy_regions.id
    reference_id         = "renewable-energy-regions"
  }
}
```

This initiative creates a policy to ensure that VMs are only deployed in regions with a higher proportion of renewable energy sources, such as North Europe. You can find more information about this approach by visiting the Electricity Maps website and examining carbon intensity data.
The second Azure Policy restricts the creation of VMs to the B-series, known for their burstable performance and energy-efficient CPU scaling.

In Conclusion

The Corporate Sustainability Reporting Directive (CSRD) is a new EU guideline that has been adopted by the European Union. In the context of sustainable software engineering, the CSRD mandates that companies report on their endeavors to diminish their environmental impact and promote sustainability. This includes initiatives related to the utilization of renewable energy, energy-efficient hardware and software, and practices aligned with the circular economy. Such reporting measures aim to enhance transparency and accountability among companies while encouraging the adoption of more sustainable practices. For publicly listed companies, compliance with these regulations is mandatory starting in 2024, with other companies expected to follow suit later. My colleagues and I are enthusiastic about assisting companies in embracing sustainable software engineering. Stay tuned for more knowledge sessions and blogs on this subject, and explore opportunities to address readily achievable sustainability goals tomorrow! </>

Understanding the Value of Value Stream Mapping

I'm willing to bet that most readers will agree that a good product, feature, or solution means that end users are able to interact effectively with it – and that this successful experience is highly dependent on the people who create the solution. This means that those creators must interact effectively with each other--and the tools and technologies they use to do their jobs--to create the solution. Effective interactions in turn, rely on organizational structures and processes acting as enablers to getting the work done, not impediments. And the only way that we can design effective interactions to deliver successful solutions is by looking at the entire system that is used to create the product or solution. Otherwise, the solution runs a high risk of not meeting customer needs. The system I'm talking about is called the Value Stream.

Author Heidi Araya

The value stream includes the people, processes, tools, information, and the technologies that people use to do work and ultimately – if all goes as planned – create value for the customer. The value stream visualizes how the work flows (rather than what the work is). Imagine a feature being delivered to customers. This feature, and all similar work types, would go through various stages of being worked on before it gets into customer hands. With value stream mapping, we would map the various stages of work that work item goes through as it is being worked on, delivered, and maintained, rather than what each feature is.

A value stream isn't a process, but is a collection of processes and interactions that start with ideation and are complete when the customer receives value from the work that was done. In this sample organization, which is an insurance company, you can see what I mean. The gray bars are the internal departments, but if a customer wants to get something done such as file a claim and get their claim paid out, many departments internally would have to interact. That's why you sometimes experience pain in your own customer experiences. It is usually due to a handoff somewhere along the way, where the company isn't optimizing for your experience but rather their own internal structure.

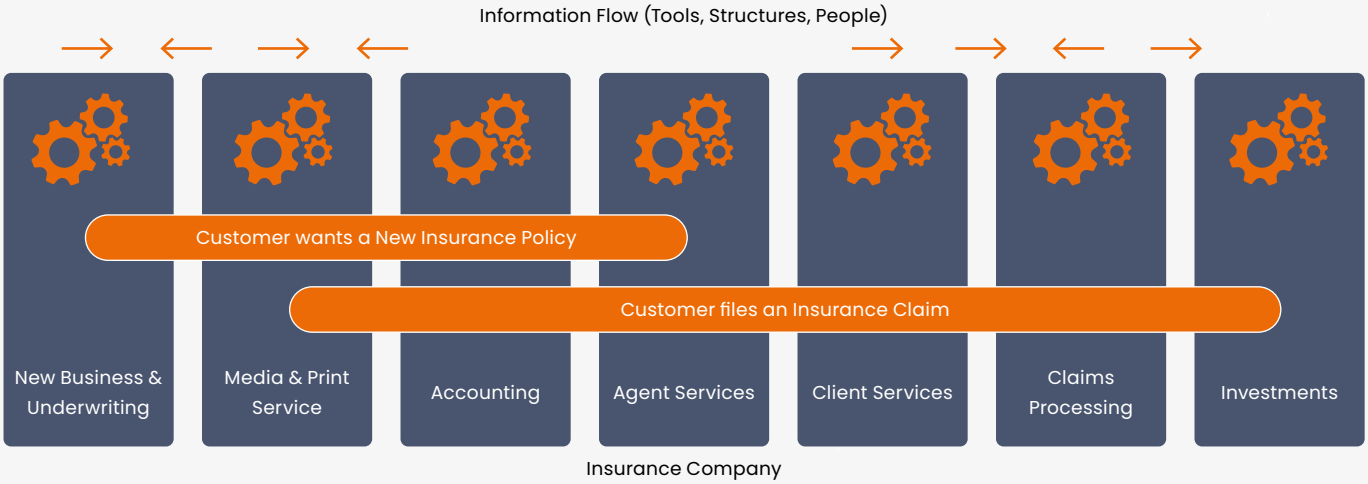


Figure 1: Value Stream Flow in Insurance Company

Understanding the value stream & working to improve it is crucial. The biggest challenges occur across the system, between departments and people, which is largely invisible today because each department has its own metrics they measure. No one is tending to the value stream or overall system metrics. This gap can cause organizational performance to suffer and people to lose sight of customer needs in favor of their own siloed success.

Benefits

Here are four ways that Value Stream Mapping can revolutionize the way you work and deliver exceptional outcomes.

1. **Making the Invisible, Visible**: Software is a realm where much is invisible. Achieving a holistic understanding of the system can be challenging due to intricate build pipelines, complex processes, interpersonal interactions, and dependencies. VSM allows us to visualize the end-to-end process, uncover hidden constraints, facts, and assumptions, and identify areas for improvement.
2. **Embracing Multiple Perspectives**: VSM brings together a diverse group of stakeholders. In a complex process, a single individual cannot possess all the knowledge of the system. By involving a variety of perspectives and people, we open the door to enabling collective intelligence to emerge. With multiple viewpoints at the table, we can uncover blind spots, challenge assumptions, and uncover novel solutions that may have otherwise been overlooked.
3. **Achieving Alignment**: As humans, we tend to think our perspective on the system is correct and rarely challenge our assumptions. Bringing people together in one place to talk through what each person notices can help counteract biases and help everyone align on what they see as challenges. And when people are aligned on next steps, we're not working against each other.
4. **Continuous Improvement**: Value Stream Mapping is not a one-time exercise but an ongoing commitment to continuous improvement. It provides a structured framework for identifying waste and optimizing the overall system. Through regular VSM sessions, we can track our progress, measure the impact of process changes, and foster a culture of continuous learning and growth within our organizations.

Results & Outcomes I Have Experienced

After doing the mapping activity and taking action on the improvements identified, I have experienced the following results:

- 7-10x improvement in time to deliver work items inside teams
- 3-12x improvement in time to deliver new features to customers
- 70% reduction in support tickets and defects
- At-risk projects delivered successfully & on time

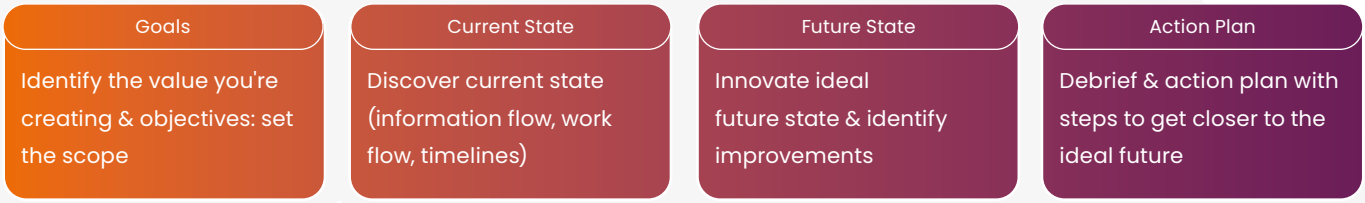


Figure 2: Value Stream Process

Hearing from our clients

At Xpirit, we might start off our engagements and assessments with an interactive workshop, tailored for the problem and engagement needs. This can take the form of a value stream mapping or process mapping workshop, where the consultants and client get together, talk through the current system and its challenges, and envision a desired future state.

This gets everyone on the same page about the current system (oftentimes the end-to-end system and interactions have never been visualized before); aligns people on the challenges they face; and then we exit with a shared agreement of the top challenges and how to solve them. This might seem elementary, but few people (if any) would have this view of the system.

The feedback I received from some recent value stream mapping sessions summarizes what many people say after this experience:

"It felt very different than anything I have ever done before. I was leery before the session, but at some point during the workshop, it began to make sense. We need to understand where we are now, before we can improve. And we need to get out of our current thinking to get to where we want to be, so that we can make real improvements to our system. I enjoyed being a part of the session and it was worthwhile."

I told him, "Yes! It's also about alignment of the problems and solutions. You have been working in this system for a while. It exists for a reason. Without context or honoring why things are the way they are, we risk breaking the system. Plus, we will be working with you in order to accomplish our joint goals. What better way to make sure we all have understood the system and agree on next steps?"

In another client, when the session was over and the people who participated were asked to provide feedback, one person said, "As a new manager, I thought that I was doing a terrible job. But now I see you all are struggling the same things that I am." This conversation sparked the idea for a manager community of practice where they could share insights and help each other solve mutual challenges.

Starting your value stream mapping journey

Value Stream Mapping can seem intimidating at first, for leadership, participants, and even a facilitator who is new to the activity! Value Stream Mapping has four main parts.

Step 1: Identify the Goals and Participants

The first step in leading and facilitating a future state value stream mapping activity is to define the scope and objectives of the activity. Identify the value stream that you want to examine and improve, define the boundaries of the process (start and end points), and set clear objectives for the activity. This is also the time to determine who would attend the value stream mapping activity.

Select the appropriate team members who have the necessary knowledge and expertise to map the process. The team should include representatives from different departments or functions involved in each step of the process.

Step 2: Map the Current State

Next, you should get to a whiteboard (physical or virtual!) and determine the various steps in the process. Create a "block" or use a sticky note for each one. There are many techniques to do this, but we suggest starting by anchoring the start of the value stream and the end; then working backwards where possible. The number of steps in a value stream are usually between 5 and 15. Use the 80% rule: don't try to map exceptions, but stick to the main goal of the value stream.

Once the flow is on the wall, review each step in more detail. Identify the wastes, inefficiencies, and bottlenecks in the process. Talk with the team about the average duration for each activity as well as the delays in between steps. You can leverage shapes in an online whiteboard, or sticky notes if you are in person.

Note: You might see complex symbols and fancy software if you look for an example online, but this isn't really required.

Once you have mapped the current state, you should analyze the information to identify some of the root causes of waste, inefficiencies, and bottlenecks in the process.

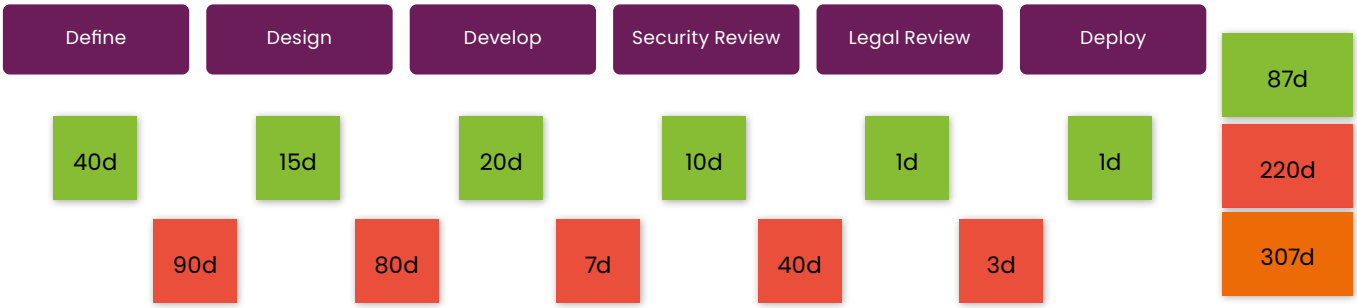


Figure 3: Value Stream Map Example

Step 3: Build the Future State Map

Using the insights gained from analyzing the current state, you can then develop a future state map that represents an improved process flow. I walk the participants through a 'happy future state' visualization exercise that imagines their current system does not exist and they can start fresh. The future state map should eliminate waste, improve efficiency, and reduce the overall duration to deliver work items that provide value to customers. Build in some feedback loops so you're able to validate that customers receive value from what's being delivered.

Step 4: Develop the Action Plan

Once you have developed the future state map, you should develop a plan to implement the changes needed to achieve the future state. This plan should include specific actions, responsibilities, timelines, and metrics to track progress. Be sure to set regular review meetings after the conclusion of the value stream mapping to ensure that regular progress is being made on the actions that were agreed.

Conclusion

Understanding and optimizing the value stream isn't just a managerial responsibility; it's a collective endeavor that involves everyone from different departments and roles. By embracing Value Stream Mapping, organizations can gain a comprehensive view of their processes, from ideation to customer value delivery. The method makes the invisible visible, incorporates multiple perspectives, aligns team members, and fosters a culture of continuous improvement.

The real power of VSM lies in its ability to break down silos, offering a holistic system perspective that single-department metrics often overlook. It redirects focus from individual success to customer-centric outcomes. The significant improvements in delivery times, reduced support tickets, and successful project completions demonstrate its impact.

Starting your VSM journey may appear daunting, but the benefits far outweigh the initial apprehensions. By diligently following the four key steps—Identifying Goals and Participants, Mapping the Current State, Building the Future State Map, and Developing an Action Plan—you set the stage for sustainable improvements.

In closing, Value Stream Mapping is more than a tool; it's a shift in mindset. It requires a commitment to scrutinizing and evolving your organization's processes continually. And as many can attest, the transformation is not just worthwhile; it's essential for any organization aiming to remain competitive and customer-focused in today's complex business environment. </>

InnerSource

Is InnerSource written weird? You will find a few related hits in Google if you search for "Inner Source", but back in the early days, no related hits where found. So, it was decided to call it InnerSource to get better reach!

Authors Arjan van Bekkum & Jasper Gilhuis

What is InnerSource?

InnerSource can be defined as the application of open-source software development principles within an organization's internal software development processes. It draws on the valuable lessons learned from open-source projects and adapts them to the context of how companies create software internally.

Similar to the familiarity of "Open Source", InnerSource encourages collaboration within the confines of an organization. It entails leveraging publicly available software, often used by developers in their daily work, and allows for feedback, including requests for new features, bug fixes, and changes, fostering collaboration akin to open-source projects.

InnerSource operates on four core principles, briefly summarized here, with more details available at InnerSource Commons[1]. InnerSource Commons is a community-driven organization that aims to promote and facilitate the adoption of InnerSource practices to improve software development within organizations. It provides a platform for knowledge sharing, collaboration, and the development of valuable resources for the InnerSource community.

[1] https://innersourcecommons.org


Xpirit

1. **Openness**: Openness in InnerSource projects ensures accessibility and simplifies contributions. It involves well-documented projects, making it easy for anyone within the organization to discover, understand, and participate. Host team contact information is readily accessible, and intentions to accept InnerSource contributions are communicated through relevant channels, promoting successful collaboration.

2. **Transparency**: Transparency is fundamental for effective InnerSource collaboration. Host teams must provide clear insights into the project's direction, requirements, progress, and decision-making processes. Communication should be detailed and accessible to individuals beyond the core team, facilitating contributions from guest teams.

3. **Prioritized Mentorship**: InnerSource relies on mentorship from host teams to guest teams, guided by trusted committers. This mentorship elevates contributors on guest teams, enabling them to engage with and modify host team projects effectively. Host teams should prioritize mentorship, assisting guest team contributors when needed, and fostering beneficial relationships within the organization.

4. **Voluntary Code Contribution**: InnerSource thrives on voluntary participation, where guest and host teams engage willingly. Guest teams contribute code to host teams and accept these contributions voluntarily. This voluntary approach ensures alignment with each team's objectives, allowing host teams to accept contributions that align with their mission and guest teams to prioritize contributions that serve their goals. Full collaboration extends to code contributions to maximize InnerSource's benefits.

Why InnerSource and the Problems It Solves

When adopting InnerSource within your organization, defining your goals and understanding what problems it can address is essential. Clarity in your objectives helps people relate to and engage with InnerSource effectively.

Are you aiming to improve Developer Velocity, as measured by the Developer Velocity Index (DVI)[2], which correlates with faster revenue growth, higher shareholder returns, increased innovation, and improved customer satisfaction? Alternatively, are you fostering a collaboration mindset, emphasizing knowledge sharing and collaboration? Perhaps your focus is on breaking down traditional boundaries through DevOps practices. Identifying your true north star for InnerSource enables you to tailor its implementation to address specific challenges and objectives within your organization.

[2] https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/tech-forward/why-your-it-organization-should-prioritize-developer-experience

Benefits of InnerSource

The advantages of adopting InnerSource are substantial and include:

1. **Mitigating Inter-Team Dependencies**: When teams operate in isolation, working solely on their individual "projects" or "repositories" without sharing their work, it often leads to code duplication across multiple areas. This results in wasted effort as people tackle the same problems independently and can also introduce subtle variations in behavior for identical solutions. InnerSource promotes knowledge sharing and collaboration on shared solutions, significantly reducing code redundancy and enhancing efficiency.

2. **Resolving Dependencies Effectively**: In larger organizations, there's typically a constant struggle for resource allocation and prioritization. This often leads to battles outside of the team's immediate focus. InnerSource helps by providing teams with visibility into available software resources and contacts within the organization. This transparency enables teams to collaborate on improving code or adding new features, all with the approval of the original owners, without waiting for prioritization decisions. While it requires some initial coordination, it is often more time-efficient than waiting for prioritization decisions.

Interaction Model

As we are building communities around projects you can clearly see communication is key. In this asynchronous world, let alone timezone differences and cross-organization collaboration, it is obvious that you need to set up your guidance in a clear and easy-to-find way. GitHub provides a comprehensive set of documented principles and practices to assist you in getting started with community collaboration. These resources cover various aspects, from establishing code of conduct guidelines and creating community profiles to utilizing pull request templates. Additionally, GitHub offers a range of communication tools to support effective collaboration within your community. You can access these valuable resources at https://docs.github.com/en/communities.

1. **Product Team**: The original product team plays a pivotal role in the development and upkeep of the core project. They are the primary decision-makers, determining which contributions to accept or reject. Additionally, they provide valuable guidance and mentorship to external contributors, ensuring the project's alignment with its goals and maintaining its overall quality.

2. **Product Owner**: The product owner defines the project's overarching vision, goals, and priorities. Collaborating closely with the original product team ensures that contributions harmonize with the project's objectives. Often, they prioritize specific features or enhancements based on user needs and market demands.

3. **Trusted Committers**: Trusted committers are individuals or team members who understand the project and have earned the community's trust. Their primary role involves reviewing and approving contributions from external contributors. Beyond this, they are crucial in mentoring and guiding contributors, ensuring the project's ongoing quality and consistency.

4. **Contributors**: Contributors are external individuals or teams that aim to make valuable contributions to the project. They actively submit code, bug fixes, or new features for review and integration into the project. Seeking feedback and collaboration within the project's community, contributors drive the project's evolution and improvement.

5. **Consumers**: Consumers, which include end-users and stakeholders, are the beneficiaries of the project's functionality. They utilize the project or product created through the collective efforts of the original product team, external contributors, and trusted committers. By leveraging these contributions, consumers meet their needs, provide usability feedback, and enjoy ongoing enhancements.

InnerSource Patterns

The InnerSource Patterns are a valuable resource that offers actionable insights and best practices for implementing InnerSource principles within an organization's software development processes. These patterns serve as a road-map to facilitate effective collaboration, knowledge sharing, and project contributions, mirroring the successful dynamics of open source communities. By harnessing these patterns, organizations can streamline their development workflows, cultivate a culture of transparency, and drive innovation through collective efforts. Each pattern provides a structured approach to address specific challenges,

making the adoption of InnerSource a well-guided and efficient endeavor. You can explore these patterns in detail at InnerSource Commons Patterns[3].

One particularly noteworthy pattern that stands out in revolutionizing workplaces through InnerSource is the "Gig Marketplace" Pattern.

Gig Marketplace Pattern

The "Gig Marketplace" pattern is dedicated to dismantling organizational silos by establishing an internal marketplace for tasks or projects. This innovative approach empowers teams to collaborate with flexibility and efficiency, offering and requesting expertise or services across different departments. This pattern encourages the free flow of skills and resources, enabling teams to tackle challenges and complete projects swiftly while nurturing a culture of collaboration and knowledge exchange.

Areas to apply InnerSource

Cloud Infrastructure

The landscape of cloud architecture is evolving and growing increasingly intricate. Notably, many companies are witnessing the emergence of Cloud Centers of Excellence (CCoE). These entities primarily shoulder the responsibility of managing shared infrastructure within cloud environments. Beyond infrastructure management, they are vital in monitoring security and ensuring its continual upkeep. Within CCoEs, teams specializing in these tasks are commonly called platform teams.

Modern cloud infrastructures often follow the hub and spoke model, exemplified by Microsoft's Cloud Adoption Framework (CAF). In this model, the hub represents the centralized component responsible for monitoring and regulating both inbound and outbound traffic. Conversely, spokes represent isolated workloads where teams can execute their software or applications. These spokes are intricately linked to the hub. Typically, it falls upon the workload teams to create and manage the specific infrastructure they require.

To ensure that workload teams adhere to compliance standards, the platform team equips them with essential building blocks for infrastructure creation. These building blocks are available to all teams needing infrastructure resources, including the platform team. Building blocks are often constructed using tools like Bicep or Terraform, both of which support the creation of modules that can be hosted in repositories such as Azure Container Registry or Terraform Cloud.

Crucially, when the source code of these building blocks is accessible to all teams, any team member can contribute changes or updates. However, for quality control and to ensure ongoing compliance, all alterations to the building blocks require approval from the Platform team.
This mechanism ensures that the building blocks continue to meet the necessary standards. In the context of InnerSource, the platform team serves as the trusted committer, overseeing these collaborative contributions.

Utilizing Packages

In the contemporary landscape of software development, packages have become indispensable. Both frontend and backend applications heavily rely on these packages. Many of these packages are open-source and meticulously maintained by passionate individuals.
They find their hosting platforms in package managers like NuGet and NPM, with GitHub as one of the most prominent platforms for hosting these open-source packages.

GitHub's foundation rests on principles of developer experience and open-source collaboration. This orientation means the source code for numerous packages is accessible to anyone, allowing for contributions from a wide community of developers. Changes to these packages undergo review and approval processes, typically overseen by package maintainers—dedicated groups of individuals who consistently contribute to the package's development and upkeep.

In addition to open-source packages, organizations also rely on company-specific packages. These packages often encompass specialized functionalities, such as authentication or logging methods. Rather than each team independently reinventing these functionalities, organizations follow a similar principle: creating packages that can be shared across multiple teams. These packages are available through platforms like Azure DevOps Artifacts or GitHub Packages.

When multiple teams within an organization use these shared packages, it becomes essential that they have the flexibility to make adjustments and improvements as needed. Embracing the same open-source principles that govern external packages, these organizations naturally foster a community of regular contributors. Within this community, individuals emerge as trusted committers responsible for reviewing and approving changes to these vital shared packages, ensuring they remain robust and aligned with organizational needs.

[2] https://patterns.innersourcecommons.org/

Applications

Like infrastructure and open-source packages, application developers can also adopt an open-source approach. Open-source applications often serve as alternatives to well known applications, for example Photoshop and The Gimp. Some companies even choose to open-source the tools they use, making them accessible to all. By doing so, they harness the collective power of the community to enhance these applications. The same principles that apply to open-source packages are extended to open-source applications, allowing anyone to contribute new features or fix bugs. Trusted committers play a pivotal role in reviewing and approving these changes.

Now, imagine if these open-source principles, championed by passionate individuals, were applied to company software. Picture making the applications within a company available for everyone, enabling all employees to contribute to the company's software.

This approach fosters collaboration among teams and departments, effectively breaking down silos and encouraging knowledge sharing. It's a recipe for innovative solutions as a broader set of eyes scrutinizes the codebase, potentially catching bugs, security vulnerabilities, or design flaws at an early stage. InnerSource encourages developers to share their expertise and best practices, elevating the overall skill level of your team and mitigating the risk of knowledge loss when employees depart.

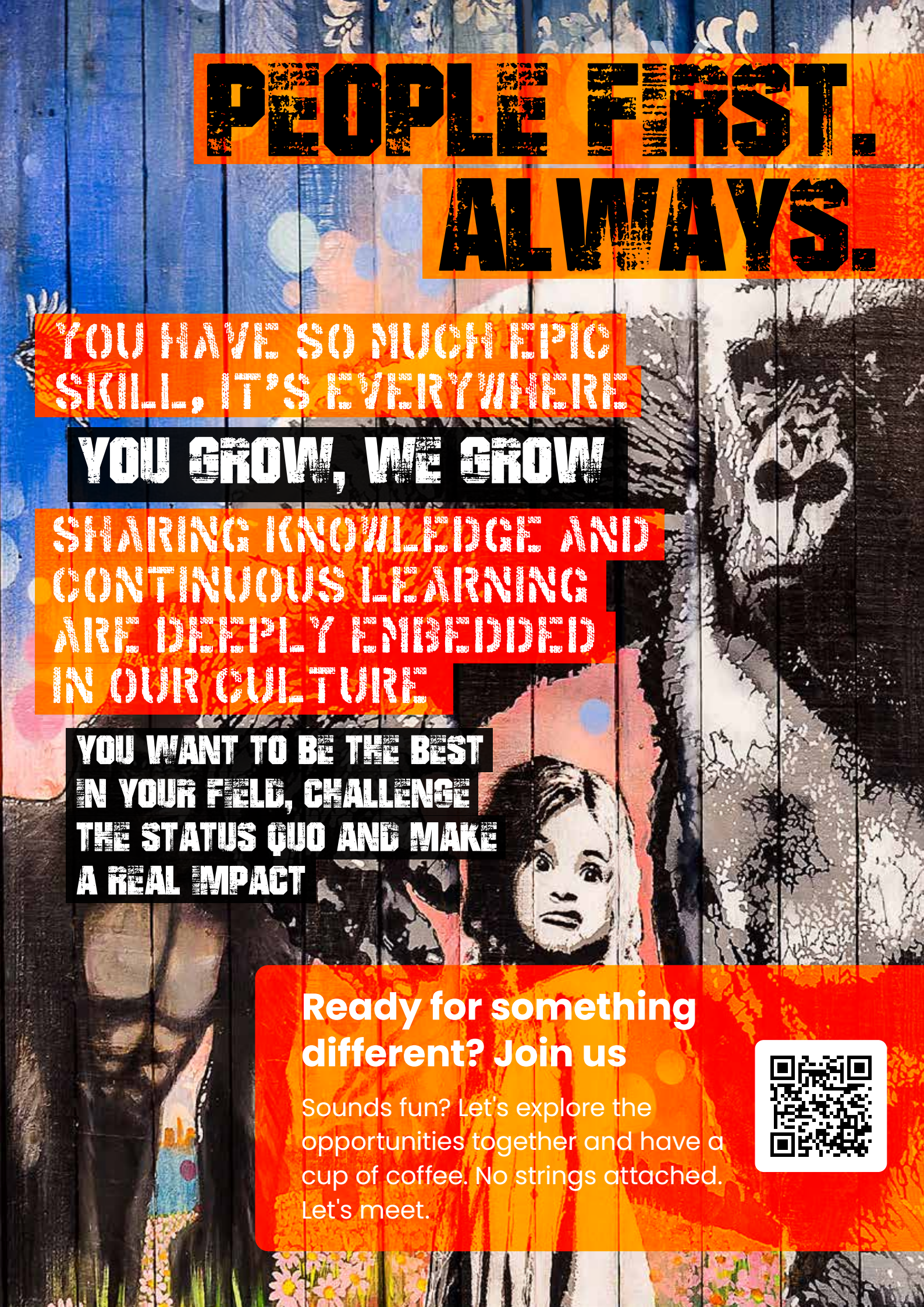
Distributing knowledge and responsibility makes the organization less susceptible to key-person dependencies. Others can readily step in to maintain and enhance the code if a developer leaves. Promoting InnerSource cultivates a culture of openness and transparency, resonating throughout the organization and enhancing company culture and employee morale.

Summary

This article explores InnerSource, a practice that brings open-source principles to internal software development within organizations. InnerSource encourages collaboration and feedback while maintaining security boundaries. It operates on four key principles: Openness, Transparency, Prioritized Mentorship, and Voluntary Code Contribution. These principles address organizational challenges such as improving Developer Velocity and fostering collaboration.

The benefits of InnerSource include reducing inter-team dependencies and resolving resource allocation challenges in larger organizations. It promotes collaboration, knowledge sharing, and efficiency. The article also outlines a role-based interaction model involving the Original Product Team, Product Owner, Trusted Committers, Contributors, and Consumers, all working together to develop and maintain projects.

Embracing InnerSource helps you build an inclusive organization, where people are able to showcase their expertise and offers a modern approach to work that aligns with the preferences and values of younger generations. It promotes flexibility, cross-functional collaboration, knowledge sharing, and inclusivity, all of which can enhance job satisfaction, innovation, and organizational agility. </>


PEOPLE FIRST.
ALWAYS.
YOU HAVE SO MUCH EPIC SKILL, IT'S EVERYWHERE
YOU GROW, WE GROW
SHARING KNOWLEDGE AND CONTINUOUS LEARNING ARE DEEPLY ENBEDDED IN OUR CULTURE
YOU WANT TO BE THE BEST IN YOUR FIELD, CHALLENGE THE STATUS QUO AND MAKE A REAL IMPACT
Ready for something different? Join us
Sounds fun? Let's explore the opportunities together and have a cup of coffee. No strings attached. Let's meet.

If you prefer the digital version of this magazine, please scan the qr-code.

www.xpirit.com