Microsoft

Microsoft Xbox team embraces SRE role to build game streaming

Overview

In the past, the Xbox Reliability Engineering and Operations (xREO) team was your typical service engineering team, spending most of its time firefighting or deploying code. That began to change in 2018, when the team was tasked with designing and building an automated CI/CD pipeline for a globally distributed Kubernetes deployment for a new game streaming system from Microsoft. With the new pipeline, cloud infrastructure and service code that would have taken six months to deploy by hand can now be deployed with a few simple commands, and adding a new Azure region to the deployment is as easy as adding six lines of code. Today, repetitive burn-downs and constant firefighting are a distant memory, and service availability has improved by an order of magnitude. Through this journey the xREO has adopted a Site Reliability Engineering (SRE) approach. The team is now a trusted part of the development team, is consulted early and often, and, with developers now able to deploy their own code, is free to focus on more proactive, higher-value tasks.

Table of Contents

The Team

Part of the Microsoft Gaming Division, the Xbox Reliability Engineering and Operations (xREO) team supports all gaming at Microsoft. The team's main job is running the Xbox Live gaming service, which has millions of active monthly subscribers in more than 40 countries and regions. Like most operations teams, the xREO team spends its time maintaining the data centers where the Xbox Live service runs, deploying new code as it's handed off by

developers, and performing all the day-to-day tasks required to keep the service up and running.

While the team's role is important, truth be told, until recently, it was also somewhat mundane. "Someone else designed and built everything—we only ran it," recalls James Whitesides, who joined the team as a service engineer in May 2017. "The architecture was fairly monolithic and rigid, with the service code coupled tightly to the underlying infrastructure, which left us beholden to some pretty strict requirements and a lot of big dependency chains. Even small changes posed a significant risk, which meant we spent a lot of our time firefighting. Our mode of operation was mostly reactive, and we weren't really empowered to do much about it."

The largest frustration among team members, however, was the constant churn of hyper-repetitive tasks. "There were lots and lots of burndowns, in which we performed rote tasks over and over again," explains Whitesides. "Without a programmatic interface into the system, everything required a lot of manual effort. We were constantly building custom scripts, most of which were used once and then discarded."

Developers who tossed their code over the proverbial fence for the xREO team to deploy had their own, related frustrations. "Even relatively small changes required a lot of developer effort, requiring them to jump through a lengthy series of hoops," says Whitesides. "There was a constant fear of breaking things, which left developers feeling pretty constrained in the code they could write."

The Challenge

All that began to change in early 2018, when the xREO team was approached to support the development of project xCloud, an on-demand cloud game streaming experience that will let users across the globe play the games they want from any screen, at any time. A few months later, in May 2018, five service engineers, Whitesides included, were formally assigned to the project.

Unlike the Xbox Live service, which runs in just a few data centers, the new game streaming system required a different approach. To minimize latency and thus ensure an optimal gamer experience, the new system would take advantage of the worldwide reach of Microsoft Azure to scale globally, with connected replicas of the service deployed to every Azure region where the service will be available—starting with a subset of Azure regions, with plans to rapidly scale that number to all Azure regions.

From the start, it was obvious that the old way of doing things wasn't scalable enough. To facilitate global deployment, the xREO team had to adopt a site reliability engineering practice (better known as SRE). That meant stepping outside of its traditional service engineering role, reinvent how it worked, and design and build a robust continuous integration and continuous delivery (CI/CD) pipeline that could:

- Support a rapid development cycle with extremely tight deadlines.
- Automate the deployment of both infrastructure and code to as many Azure regions as desired, ensuring that every region is provisioned identically.
- Bind all of the microservices running in each region together and link the regions to each other.
- Deliver a global view of both service health and underlying Azure resource usage.
- Remain code-agnostic, so that developers can use additional Azure services if desired.

The Journey

> *We had some automation, but it was rudimentary and disconnected*
>
> JAMES WHITESIDES
> Program Manager, xREO Team

At the time the xREO team became involved, other than running on Azure, only one major architectural decision had been made: the service code would be containerized, as a means of loosely coupling it from the underlying infrastructure. Through the adoption of containers, with developers owning all the code inside the containers and xREO team owning all the infrastructure outside the containers, clear boundaries between the teams were established, dependencies were removed, and collaboration would be easier. Containers would also make the new system far less rigid, thus avoiding many of the limitations that both developers and service engineers had faced in the past.

"Containerization was an obvious choice," says Whitesides. "Docker containers would let developers code the way they want and call the services they want to use, without having to know the details of where their code runs."

Having decided on Docker, the next step was to tackle container orchestration. Again, the decision was straightforward: Kubernetes, the de-facto industry standard, was the obvious choice. Having access to Azure Kubernetes Service, a fully managed Kubernetes service, made

Kubernetes even more compelling, as it would eliminate the need for the xREO team to manually manage VMs and Kubernetes clusters.

"Building and running our own Kubernetes clusters would have been too labor-intensive, requiring a much larger team," says Whitesides. "However, with Kubernetes comes an increased reliance on open source. Fortunately, .NET Core runs on Linux, so developers of the service could keep coding in C#. So far, up to this point, the choice of technologies was all pretty obvious."

> *We decided to provision and manage all deployed assets, including both code and infrastructure, through a source-controlled environment*
>
> JAMES WHITESIDES
> Program Manager, xREO Team

The first major joint decision came next. "We decided to provision and manage all deployed assets, including both code and infrastructure, through a source-controlled environment," explains Whitesides. "Developers were already using Azure DevOps for source control; it made sense to expand that same environment to define the necessary infrastructure in code as well—and take the first step toward an automated, more DevOps-centric CI/CD model."

At this point, developers were still somewhat ambivalent to the efforts of the xREO team. "Their attitude at the time was, 'as long as everything works when we check in our code, we don't have a problem with it,'" says Whitesides "Lots of developers are dabbling in Kubernetes, but it gets more complex in production, which is where our team's work really began."

Developers were already writing code, so the xREO team didn't have the time to automate everything up-front. Having to do everything manually, the small team quickly became overwhelmed by the number of tasks and amount of effort required to build each Azure Kubernetes Service cluster. It took 2-3 days for initial provisioning and configuration, 4 days for resource creation, 1-2 days for connectivity, and 1-2 days for troubleshooting—resulting in about 2 weeks of work for one regional deployment.

"When we started deploying clusters, there were only 12 microservices," recalls Whitesides. "Developers then came to us with a revised architecture consisting of 30 to 35 microservices, with complex interconnectivity requirements, as well as additional requirements for regional mapping. At the rate we were going, with only simple scripting and minimal automation, it would have taken our small team of five about six months to deliver the first set of regions, along with the necessary test environments."

To make matters worse, the resulting environment wouldn't have been all that different than the one they had been supporting for years. "At that point, nothing was configurable and there was no CI/CD, which meant that it was still pretty rigid and fragile," says Whitesides. "And this new environment was even more complex than the old one, which would have made it even harder to maintain."

Faced with an impractical amount of work, the team began to focus on the lack of repeatability in its existing processes. "We had some automation, but it was rudimentary and disconnected," explains Whitesides. "Developers didn't all define their services the same way every time, tooling was haphazardly integrated, and monitoring was inconsistent. Clearly, we needed something better."

The team's first push toward automated resource deployment and configuration involved the use Azure PowerShell scripts and Azure Resource Manager templates to rapidly and repeatedly provision Azure resources. However, after those resources were deployed, the team still had to bind them to the platform and ensure secure service-to-service communication.

"By this time, back-of-the-napkin estimates called for the provisioning of 1,200 such resources across the initial target number of Azure regions," says Whitesides. "Scaling deployment was still going to be difficult unless we could find way to declare all our connections when we provisioned the assets. While we could save these secrets in a config file and add it to the service's build code, or worse, add the secrets directly to the build, this approach required human interaction with secrets, manual editing of files, and a rebuild of the service for any new region—all of which violated our secure design requirements."

The team decided to use Azure Key Vault, which gave them a secure and scalable way to manage those secrets separately and bootstrap them into the platform at runtime. "With key vaults, developers would no longer need to care where the code would run; they could simply call the same secret name, knowing that it would map to the specific region into which it was provisioned," says Whitesides. "They liked how this would simplify coding and eliminate the need to write a wrapper for each regional deployment."

> *Today, for a lightweight service, this deployment time is less than half an hour.*
>
> JAMES WHITESIDES
> Program Manager, xREO Team

Next, the team tackled deployment of the service code itself. "To install service bits, developers run builds in Azure Pipeline that push the images of those service bits into Azure Container Registry, which is bound to the deployment through variables defined in an environment configuration," explains Whitesides. "To bring up a brand new service or push an existing one to a new region for the first time, developers simply declare it in a .cmd file that triggers a deployment of the resources and setup required for that service in the new location. Today, for a lightweight service, this deployment time is less than half an hour."

With the ability to deploy a functioning Azure Kubernetes Service cluster to as many regions as desired now fully automated, the final step was to deliver a traffic management solution for multi-region connectivity. "Mapping a single region to its global and region specific endpoints while load balancing everything correctly was the big challenge," says Whitesides. "Azure Traffic Manager gave us the flexibility to implement exactly what we needed. It allows our developers to declare the type of routing they want, where to host that routing solution, and how the client interacts with the Azure Kubernetes cluster—all in a really easy, straightforward way."

The Results

Ten months after it joined the effort, the xREO team finished its automated deployment pipeline. Today, that pipeline is being used to deploy more than 35 Azure Kubernetes Service-based microservices that rely upon more than 100 resources (per-region) to numerous Azure regions. Plans call for expanding to additional Azure regions within months, and eventually to every region. "To deploy a new region, we add six lines of code to a configuration file and wait for resources to spin up," says Whitesides. "Extending the architecture to employ new Azure services is just as easy."

> *Today we can be proactive instead of reactive—focused on new tooling instead of fixing problems*
>
> JAMES WHITESIDES
> Program Manager, xREO Team

Compared with where they started, the team's work has made a remarkable difference in the global scalability of new game streaming system. "Coming into the project, our small team was looking at six months of full-time effort for an initial deployment to all target regions," he recalls. "Today, if needed, we could delete everything in every Azure region, and then redeploy it all from scratch in just seven days, to as many Azure regions as desired—with most of the time spent waiting for scripts to run."

With developers working on the system now largely self-sufficient, the xREO team is free to focus on further architectural improvements. “We’re already starting to tackle more SRE-centric tasks, such as the effective monitoring of thousands of Azure resources, further simplification of Azure Kubernetes Service management in a global multi-region model, and building core validation into all deployments managed through our code,” says Whitesides. “We’re still involved with day-to-day service engineering, but it’s no longer an all-consuming series of mundane tasks—even though we’re now at global scale.”

In addition, service availability is extremely high. “The entire environment is all very stable and hands-off,” says Whitesides. “Decoupling the service code from the infrastructure through containerization has been the largest contributor, but the CI/CD pipeline we’ve built has also enabled us to do things like service delivery optimization, green/blue deployments, and canary deployments”.

As an added bonus, for the xREO team, repetitive burn-downs are now a distant memory. “Today we can be proactive instead of reactive—focused on new tooling instead of fixing problems,” says Whitesides. “Our relationship with the development team is also different. We used to just be the people who kept the lights on and occasionally fought fires, which created a culture that wasn't conducive to the tenets of site reliability engineering that we strive toward. Today we’re a trusted part of one larger team, are consulted early and often, and are even invited onstage at various conferences to talk about our work.”

Resources

Tools and technologies

DevOps solutions on Azure

Azure Kubernetes Service (AKS)

Azure Pipelines

Azure Resource Manager

Azure Key Vault

Azure Traffic Manager

Azure Container Registry

Online courses

Evolve your DevOps practices

Introduction to Azure Kubernetes Service

Explore more DevOps Stories >