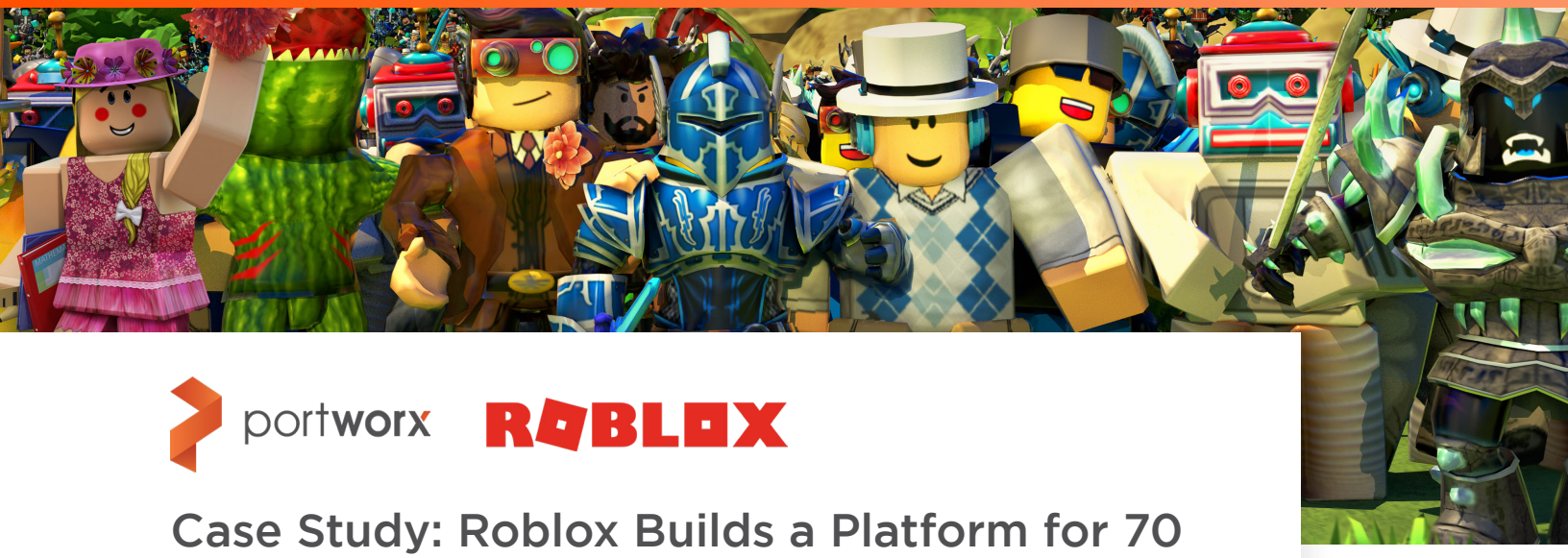

portworx ROBLOX

Case Study: Roblox Builds a Platform for 70 Million Gamers with Portworx and HashiCorp Nomad

CHALLENGES

- Maintaining availability and performance of a platform used by 70 million gamers worldwide while keeping operations costs in check
- Preventing data loss in the case of large-scale failures
- Scaling a large data platform without an over-reliance on additional headcount

SOLUTION

- HashiCorp Nomad for managing vm- and container-based workloads scaling to 1000s of nodes
- Portworx PX-Enterprise for cloud native storage and data management

RESULT

- With Portworx, stateful applications like databases are as easy to manage as stateless services without the need to sacrifice critical resiliency and redundancy.
- Portworx support and engineering teams provide critical expertise to the Roblox team, allowing their platform to scale to millions of users without having to operate and scale a storage platform.
- Now in the case of an attack or other large scale failure, Roblox can easily recycle and rebuild their entire environment within one hour or less.

KEY TECHNOLOGIES DISCUSSED

Infrastructure – AWS, Azure, on-premises (bare metal)
Container Runtime – Docker
Orchestration – HashiCorp Nomad
Stateful Services – CockroachDB, MongoDB, InfluxDB, ElasticSearch, GitLab, Jenkins, Docker Registry

Roblox is an insanely popular gaming and entertainment company with 70 million active monthly players. We sat down with Rob Cameron to learn about how Roblox is leveraging containers to run stateful applications at scale. This is a transcript of that conversation.

Rob Cameron
Principal SRE at Roblox

Can you tell us a little about Roblox?

Sure. Roblox's mission is to bring the world together through play. Every month, more than 70 million people around the world have fun with friends as they explore millions of immersive digital worlds. All of these digital worlds are built by the Roblox community, made up of over 4 million creators. There are many different facets to the Roblox ecosystem. So whether you're using it for social interaction, or as a gaming entertainment system, or from a STEM learning growth perspective, or learning to code, there's something that Roblox can provide for you.

What is your role at Roblox?

I'm a Principal SRE, inside of the production engineering group. That means that my job is to ensure that we're using technologies and services that guarantee that the Roblox platform runs continuously, providing value for all of our players and anyone who chooses to engage on the platform. We also continuously modernize our platform, as Roblox has grown and evolved dramatically since the company was first founded in 2004. Because of that, our strategy and how we manage everything as it relates to infrastructure comes from that legacy approach.

I was brought on specifically to focus on the orchestration side of things for the platform. We need to roll out features faster and deliver a platform that enables our 4 million creators to more easily do their jobs, all while scaling for enormous growth. Containers are definitely a part of that.

One of the biggest challenges we had to overcome was that we run one single global platform. That means that any player, anywhere in the world, whether in China, in Australia, or in United States, can interact with one another. Some gaming platforms utilize a sharded environment, which restricts you based on your geolocation but we don't do that.

We think it creates a better gaming experience, but it also creates some major technical challenges.

Can you tell us how you are using containers at Roblox?

Sure. To understand how we're using containers, you need to understand a little about our basic setup and history. Our infrastructure model is divided between what we call data centers and "points of presence" or POPs. A data center is a place that we locate databases and other types of centralized services. A POP includes routing connectivity to internet providers, as well as our own backbone, and also game servers.

Previously, all of the Roblox game servers ran on Windows. The number of game servers fluctuates based upon traffic, spiking when kids are out of school. It costs a lot of money to run those servers on Windows. The long-term risk as the platform becomes more popular is that we would have to keep writing that check. That's money we would rather invest internally to be able to provide a better user experience for players, more engineers, international expansion, etc.

So the decision was made to port over the game engine into Linux. That was completed in the fall of 2018, and then I was brought on board to help operationalize and manage the Linux side of things. Once in that Linux environment, we needed a way to manage at scale all the different services that were part of the platform, and automate deployments, etc. And that's where containers come in.

Since we still have some services running on Windows and some that are not running in containers, we opted for HashiCorp Nomad as the orchestration engine because it can do both Windows and Linux containers and VMs.

“

Since we still have some services running on Windows and some that are not running in containers, we opted for HashiCorp Nomad as the orchestration engine because it can do both Windows and Linux containers and VMs.”

Initially, we started out using containers for the game server side only. We then started thinking, “what else could we put in containers?” We looked at doing various other microservices such as GraphQL endpoints for example or service discovery and dynamic load balancers and various infrastructure tasks. Once we had done that, we also migrated a lot of our network monitoring over to containerized services instead of running on one or two different point hosts.

That has gone really well for us, because we built a steel thread by focusing on our game servers. We built all the infrastructure that allows us to use containers, and now we’re looking at running microservices for our platform including various in-memory databases, and other container-friendly databases.

We want to eventually migrate everything over into containers except some of our very high-performance databases. But what we don’t want to do is hamstring development teams by forcing them to run things in containers where it doesn’t make sense. We want to have people use the best possible tool for the job. And some teams are more open to adopting to containers than others. Overall I think we’ve been very successful with our container implementation.

Currently we’re running around 50 nodes in our Nomad cluster, but the idea is the game servers alone will be around 5000 to 10,000 nodes. So as we expand the platform and move other services over into Nomad, that would continuously grow and we will eventually run an additional 2000 or 3000 nodes inside of Nomad.

“

Currently we’re running around 50 nodes in our Nomad cluster, but the idea is the game servers alone will be around 5000 to 10,000 nodes. So as we expand the platform and move other services over into Nomad, that would continuously grow and we will eventually run an additional 2000 or 3000 nodes inside of Nomad.”

What were some of the main challenges you needed to overcome in order to run stateful services (like databases) in containers?

Not surprisingly, it was persistent storage. None of the container schedulers have a great, native storage experience. But having a solution here is critical.

In my previous job at another company, the biggest issue we had was that we didn’t want to use a storage layer because we felt that per node storage or local volumes were the best because of cost and flexibility. The truth is, you can probably make it work, but if I have a database that I need to make sure never fails, it just seems way too risky to run it like that.

The cost-saving and flexibility you think you are going to get evaporate in the extra management time you have to put in.

Even with something like CockroachDB or MongoDB, which are very good at sharding and being able to have replicas of data, what do you do in your container solution if somebody just goes, "Stop!" on your task and then your directories are all garbage collected and you lose all your data? It's a big sad face, right? And I don't wanna have that sad face with data.

Even with something like CockroachDB or MongoDB, which are very good at sharding and being able to have replicas of data, what do you do in your container solution if somebody just goes, "Stop!" on your task and then your directories are all garbage collected and you lose all your data? It's a big sad face, right? And I don't wanna have that sad face with data."

Our first attempt to solve these problems in practice was when we tried to run GitLab on Nomad. Every six months we have an internal event called "Hack Week," where everyone is allowed the liberty to do whatever project they want... within reason. Some people make games, some people make tools for games, some people who are infrastructure nerds like me make infrastructure stuff. So during that week we had a team that was looking at our massive source control management problem, where we have multiple systems and multiple tools to run.

We thought, "It would be great if we could run a tool that would be easy to scale." And so we'd looked at GitLab or GitHub Enterprise. And we're like, "Yeah, but we don't really have a place to run it, we don't have dedicated VMs that we could use." So we're like, "Hey, let's take a look at running that on Nomad in containers."

We thought, "It would be great if we could run a tool that would be easy to scale." And so we'd looked at GitLab or GitHub Enterprise. And we're like, "Yeah, but we don't really have a place to run it, we don't have dedicated VMs that we could use." So we're like, "Hey, let's take a look at running that on Nomad in containers.""

But the problem with Nomad or any container solution is that we don't necessarily have a consistent storage layer. We could of course rely on local node storage, but then we have to worry about managing that. So, we looked at a few different solutions like GlusterFS doing NFS shares or Ceph but decided against it because we didn't want to have to actively manage our storage like these tools require.

Ultimately, which data layer to go with just comes down to how much responsibility you want your organization to take over your data. Gluster, Ceph (or some others) are widely used solutions that exist in various distributions that seemingly are easy to use.

However, it's risky, because what do you do when the data is gone or that system fails? Do you have a big enough team to help and support manage it? Do you have enough people to focus on backups, restoration, and the operational lifecycle? If you do, maybe that's the route you want to go with.

But for us, we're a fast growing company. We're doing great, but still we don't have an infinite amount of people. So we could go with those solutions, but then what do we do when it fails, and who do we trust?

For me, I'd rather work with a company like Portworx, not as a typical vendor-customer relationship, but as a partner so we can grow together, help each other out, ensure that that data is there. And if I know there's an issue, I know Portworx is gonna allocate a bunch of their engineers to get me safe and secure, and make sure most importantly that my player data isn't lost.

But for us, we're a fast growing company. We're doing great, but still we don't have an infinite amount of people. So we could go with those solutions, but then what do we do when it fails, and who do we trust?

For me, I'd rather work with a company like Portworx, not as a typical vendor-customer relationship, but as a partner so we can grow together, help each other out, ensure that that data is there."

And if I know there's an issue, I know Portworx is gonna allocate a bunch of their engineers to get me safe and secure, and make sure most importantly that my player data isn't lost."

With other solutions, that may not be something that I get. And they may not be ideal for what I'm trying to do since they are optimized for workloads running in VMs. But, our singular use case and our future use case is focused around container-based services. And that's where Portworx is a perfect natural alignment for what we're looking to do.

So with that in mind, the initial use case we purchased Portworx for was for each POP to be run independently, and we deploy Nomad clients there that are managed by central servers in our regional data center. And then on top of that, install the TICK Stack for doing telemetry and alerting and an ELK stack for centralized logging for all the different game instances. This is perfect for a containerized environment because in the case of the game servers, we just want to throw them away and refresh with a new instance but we still need to preserve those logs.

In case of an attack or in the event that a POP is breached, we can easily recycle and rebuild that entire environment including all of the game servers within an hour or less.

And so by using Portworx, we're able to run those stateful applications similar to how we would run a stateless environment but still provide the resiliency and redundancy that we need.

“

And so by using Portworx, we’re able to run those stateful applications similar to how we would run a stateless environment but still provide the resiliency and redundancy that we need.”

The initial setup of Portworx took two only hours. We had our development cluster up and running, and boom, we were able to launch tons of different stateful services. To test, we went with GitLab and then tools like Jenkins and CockroachDB. We tested all of these different things, and it was a very easy solution to use. Not a lot of effort to set it up, and the reliability was really strong. We did performance tests with three replicas for the volumes where we would take one node offline, then take two nodes offline, and see if the service would automatically failover and still keep running. In the end, it just worked great across the board.

As a result of these tests, we began to think that instead of buying a bunch of storage that we might not need on all our servers, we could instead move over to a Portworx-type of environment where we could have different classes of servers, with and without disks, and let Portworx and Nomad place the workloads for us. And we liked the idea of how Portworx has specific tuning for each container volume. That way if we want to run a database workload, we can use one type of volume which is Read-Write-Once and performance-based for the database, and for other environments, we may want to look at doing something where there’s a shared volume. For instance, if we wanted to run a local Docker registry, we could have a shared volume where multiple nodes read and write from the same volume.

The thing that really sold us was the ability to do encrypted volumes, specifically for highly sensitive data that may have player data inside of it as well as the easy ability to do snapshots and centralized reporting and management with the Portworx Lighthouse user interface.

Portworx is an important part of what we need to do to simplify our infrastructure.

“

The thing that really sold us was the ability to do encrypted volumes, specifically for highly sensitive data that may have player data inside of it as well as the easy ability to do snapshots and centralized reporting and management with the Portworx Lighthouse user interface Portworx is an important part of what we need to do to simplify our infrastructure.”

You’ve been through a lot in this project. What advice would you give to someone who is also looking at running stateful services and containers?

I would say when you’re evaluating running services in a container, you really have to understand what the desired running state of that service is in production. Something like NGiNX is easy, but more complex services, especially stateful services are different. When you’re running a complex service in a container, it’s not like running in a VM where things are very long lived and don’t move around a lot.

You have to be very clear about what you expect out of your service when things are running smoothly and also when things are failing. You should take the time to document how you expect that containerized service to operate because how you perceive how a container should run might be different from other people.

Take your time to level your expectations around what you think you're going to get out of your service, document the desired states and how you feel it should run, and then definitely take the time to test and evaluate what those states are.

For stateful services, think through things like:

1. How do I provision my volumes?
2. How do I maintain that they will continue to operate even when a server or a disk fails?
3. How do I back up, how do I restore, and what are the tools I have in-between?

For stateful services, think through things like:

1. How do I provision my volumes?

2. How do I maintain that they will continue to operate even when a server or a disk fails?

3. How do I back up, how do I restore, and what are the tools I have in-between?"

And then talk with someone like Portworx or your storage provider to ensure that you're following best practices because honestly, whatever you think you're coming up with for the first time, someone has probably run a similar use case in their environment and the more information you can gather from those existing use cases, the better prepared you are to run that service in production and run it at scale.

LEARN MORE

Portworx is the cloud native storage company that enterprises depend on to reduce the cost and complexity of rapidly deploying containerized applications across multiple clouds and on-prem environments.

With Portworx, you can manage any database or stateful service on any infrastructure using any container scheduler. You get a single data management layer for all of your stateful services, no matter where they run. Portworx thrives in multi-cloud environments.

Contact us to find out how Portworx's unique storage solution can deliver the availability, performance, and features necessary to minimize stateful application operations costs and improve revenue growth -

https://portworx.com/request-a-demo

portworx.com
www.roblox.com

@portwx
linkedin.com/company/portworx

portworx

4940 El Camino Real, Ste 200, Los Altos, CA 94022
Tel: 650-241-3222 | info@portworx.com | portworx.com