# Case Study: New Context Securely Moves Enterprises Over to Containers and Microservices with Portworx

## CHALLENGES

- Large clients require the agility of container-platforms but they cannot refactor their applications from the ground-up
- Container management platforms like DC/OS provide automation for applications but have limited support for stateful services like databases
- Solutions to connect enterprise storage to the container are fragile and don't scale

## SOLUTION

- Mesosphere DC/OS container management platform for scalable, enterprise applications
- Portworx Enterprise for cloud native storage and data management

## RESULT

- Portworx provides enterprise-class storage and data management enterprises require for container workloads, without requiring them to refactor their applications
- Portworx integrates with Mesosphere DC/OS to provide persistent storage for data services, enabling automation required to keep applications available
- Portworx provides a reliable and scalable storage and data management layer for enterprise applications which keeps New Context clients happy

## KEY TECHNOLOGIES DISCUSSED

Data Center – On-premises, AWS

Container Runtime – Docker

Scheduler – Mesosphere DC/OS, Marathon

Stateful Services – Cassandra, Consul, DiskCache, EHCache

**New Context is a lean security consulting firm that helps enterprises implement containers and microservices.**

**We sat down with Danny Purcell, Senior DevOps Engineer at New Context, to learn about the challenges enteprises face when implementing databases in containers. This is a transcript of that conversation.**

## Can you tell our readers a little bit about what New Context does?

New Context is a lean security consulting firm. We help customers with software development, and architecture and infrastructure management, often as a part of their move to the cloud. What makes us unique is that we do this from a security perspective. We really have a security-first mindset. So while we're working with a customer on, say, moving to microservices running on DC/OS, we'll see things that could be done better from a security standpoint, and try to bring those up to first-class concerns with our clients.

We are also heavily automation focused. For instance, we see clients where testing is done, but it is done manually and it would be spotty. As we work with them, we'll help them build automated testing in their CI/CD pipeline. What is really cool is that once you help get that kind of organization habit going, it becomes a natural part of their workflow, and it just keeps going. So we look to create that kind of organizational change in the companies that we work with.

## Can you tell us a little bit about your role at New Context?

I've been at New Context for a year now and primarily work with tools like Terraform and Chef setting up container platforms such as DC/OS. As a Senior DevOps engineer, I am part of the on the ground engagement with the client, really working through all the technical details that come up. That engagement could either be a project given to us where we form a team around achieving some set goal for the client, for instance getting a CI/CD pipeline going for the whole development environment. Or we'll do what we call staff augmentation which is where we will have our people integrated with the client's team.

## Can you tell us a little about a recent container project you worked on with a client?

Sure. A lot of customers we work with want to move to the cloud. But as part of that, they want to become more efficient in their software development processes so they can be more competitive. Almost invariably this leads to a platform as a service implementation, something like DC/OS or Kubernetes, where in the end, the customer gets a platform that their developers can use to more easily build, ship and run applications.

That is great for stateless, ephemeral services, you know those micro-service architectures where none of the containers actually need persistent data, but one of the first things that we run into is that there will be at least a few services that do need to keep persistent data.

**“The problem that we've seen implementing these container-based platforms is that most large organizations typically don't want to rewrite their whole stack to be micro-service oriented right off the bat.”**

The problem that we've seen implementing these container-based platforms is that most large organizations typically don't want to rewrite their whole stack to be micro-service oriented right off the bat. So we end up with a number of applications that need to be moved to Docker containers but they have particular performance requirements like disk IO or they need persistent access to temp files or use some kind of disk caching mechanism, etc.

Also typical is that they have a database as part of the stack which clients want to run on the container platform as well.

When you have a scenario like that, it gets sticky very quickly on most container platforms because the easy, happy path is just for ephemeral things. With containers, you can do fun things like move containers when things fail. The platform will have a point of view built in that says, “Hey, we’re just going to shut things down as needed and move things around and your apps will have to deal with that.”

This is probably the chief way that containers differ from the VM environments that people have been working with for a long time now. With VMs, they are typically big enough and expensive enough to set up that you don’t tend to want to move all the stuff around a lot. As a result, you can bet on your data services being there for a good long time. You tend to do more careful in-place upgrades, on-the-box changes, backup/restore and things like that in an effort to keep things going where they currently live. In container-land, it’s easier to move towards immutable infrastructure and things tend to move around and get restarted a lot more. So this is where Portworx can really come in and save the day because we have a storage fabric that integrates with container workflows, but still provides persistence as we’d expect in a VM environment.

**“So this is where Portworx can really come in and save the day because we have a storage fabric that integrates with container workflows, but still provides persistence as we’d expect in a VM environment.”**

You can still back things up. You can still have a server die and not lose your data. What is nice is all that is at the container level. And we can integrate with a number of sources of storage like NFS and EBS volumes in AWS. This makes it easier to do hybrid cloud deployments, something that otherwise would be really difficult because no container platform really supports the stateful case well.

Getting into specifics, DC/OS is probably the furthest along when it comes to support for stateful services but there are still some issues that we needed to overcome.

First, the integration with REX-Ray, while a valiant effort, doesn’t cover the use cases that our clients have. There are several problems. One is that with this tool, all volume sizes are fixed at the creation of the DC/OS cluster so we can not set per-app volume sizes using Docker containers. Portworx does not have this limitation. Portworx also supports in-line volume specification which allows our DC/OS platform users to launch containers that use storage without having to log a ticket and wait for someone to manually create a volume for their app. This is a big help for agility because the app teams don’t have to wait on manual processes to test their deployments.

**“First, the integration with REX-Ray, while a valiant effort, doesn’t cover the use cases that our clients have.”**

Also, when using a REX-Ray volume, Marathon apps are limited to one task at most. This constraint makes it very difficult to realize the benefits of a container platform for stateful apps. One of the biggest benefits of a container platform is the ability to easily add more instances to a running application so we can smoothly keep up with incoming user requests. This is known as app scaling and is part of the Marathon UI. An operator can easily increase the desired number of instances from 1 to 2, for example.

If the app were using a REX-Ray volume though, that request would be blocked by the platform since the use of a REX-Ray volume limits the number of tasks an app can have to 1. We can "scale down" to 0 to effectively pause the app and then scale back up to 1 but we can not go past that when using REX-Ray.

There are a lot of use cases where more than one instance of a stateful app is desirable. One use case would be running Consul on Marathon. Things like DiskCache and EHCache would also scale volumes with compute if run on Marathon. Then there are the occasional applications and services that, for one reason or another, make use of local storage. When an organization makes the move to containerize, the attending challenges can be exacerbated by the need to redesign various apps and services to avoid using local storage.

REX-Ray volumes are also restricted to being mounted in only one task at a time so they cannot be shared across tasks. On the other hand, Portworx volumes support a "shared" flag which, if set, allows multiple tasks to mount the same volume. So far as I've seen, Portworx is really the only one that's tried to tackle that problem.

Another closely related use case is trading off mounting a single volume between multiple containers. While REX-Ray technically supports this case, it's quite slow with EBS volumes and we have seen the EBS volumes get stuck dismounting while trying this.

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.

**"With Portworx, there is no requirement to dismount an EBS volume when replacing a task so tasks are not only free to move around but those movements can be done faster since we don't have to wait on the EBS volumes to move."**

The last problem we have had with REX-Ray is that failover works but only as long as the underlying volume dismounts properly so it can be re-mounted to another host when Marathon starts a new task for the failed app. This operation can be flaky though, sometimes it can get stuck.

With Portworx, there is no requirement to dismount an EBS volume when replacing a task so tasks are not only free to move around but those movements can be done faster since we don't have to wait on the EBS volumes to move.

## What advice would you give to someone else who was thinking about implementing stateful containers?

Start small. Don't try to do it all at once. I really like the advice of John Fiedler, the senior director of engineering at SalesforceIQ. He did a talk at a Dockercon 2015 on how to successfully run Docker in production. Much of the advice in that video is still relevant now, in particular to running stateful services. The general thrust of the talk is "don't try to do everything at once." Make it a migration path. There's very often in the software industry this desire to do a clean cut each time and start green field on things, but it is much simpler to take this migration in pieces.

As far as which stateful service to start with, the characteristics you're looking for is a stateful service with the least complex deployment model that you can get. For example, if you do something like a ring server, like Cassandra, or anything that mirrors a gossip protocol, or a ring style deployment, those tend to be a bit simpler, at least in the deployment style. Simple is a relative term when you're talking about databases and clusters of databases of course so it really depends on your application but the biggest thing is just to start small and learn as you go.

## 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.newcontext.com

@portwx
linkedin.com/company/portworx