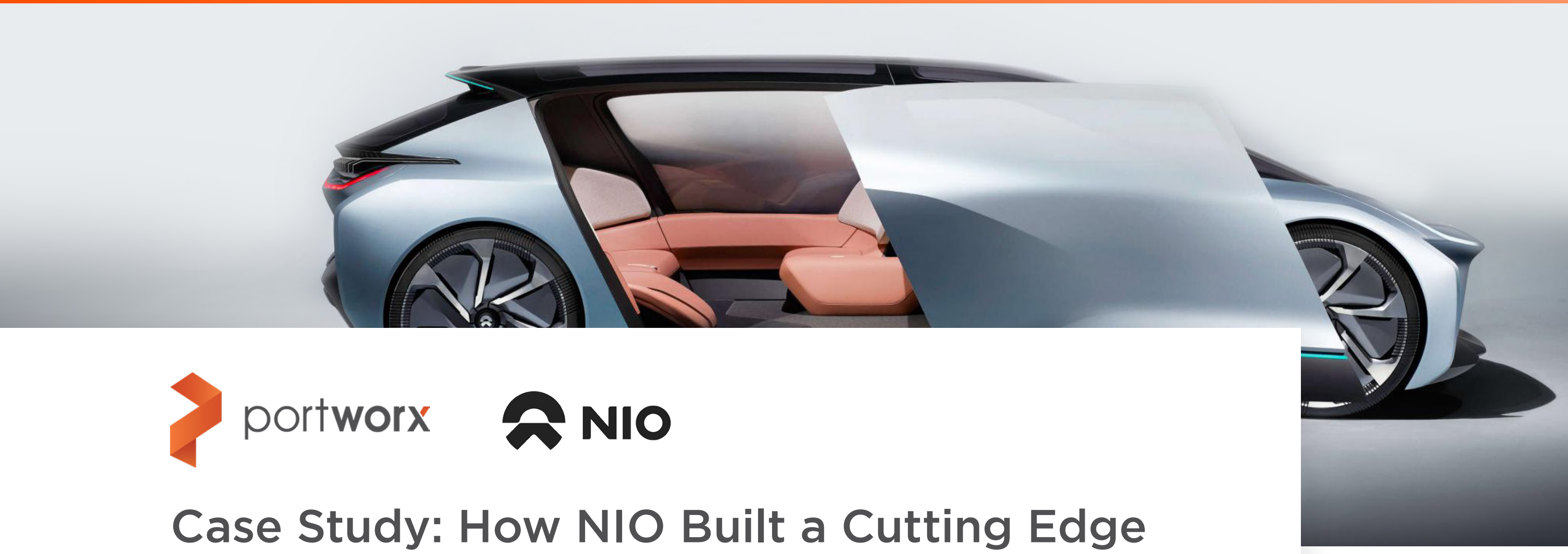

Case Study: How NIO Built a Cutting Edge Autonomous Vehicle Platform on Top of Containers

CHALLENGES

- Scaling compute and storage infrastructure independently as each autonomous vehicle fleet comes online
- Storage and data processing for up to 24 TB of data created by each vehicle per day
- Containers running on bare metal are the only way to isolate workloads while maintaining performance, but data management in Kubernetes and DC/OS is immature

SOLUTION

- Portworx provides a flexible, automated data management layer for apps running on Kubernetes and DC/OS
- NIO uses commodity servers for storage to achieve near bare metal performance without sacrificing automation enabled by containers
- Portworx provides dynamic provisioning, HA, snapshots, backup, and encryption for databases, machine learning and deep learning scenarios

RESULT

- NIO rapidly scales compute and storage infrastructure independently to meet ramped up vehicle production
- NIO takes advantage of the density, resource isolation, and portability of containers, without sacrificing crucial storage and data management capabilities

KEY TECHNOLOGIES DISCUSSED

Container Runtime – Docker
Scheduler – Kubernetes, Mesosphere DC/OS
Stateful Services – Cassandra, Kafka, HDFS
Infrastructure Provider – On-premises

We sat down with Satya Komala to learn about how NIO is leveraging containerized infrastructure to manage and store impressive amounts of data on a hourly basis. This is a transcript of that conversation.

Satya Komala
Head Of Autonomous Vehicle Cloud & Enterprise Architecture at NIO

Can you tell me a little bit about what NIO does?

NIO is a mobility company. We're working on building safe, autonomous, electric vehicles. The goal of the company is to make life better for people by giving them their time back. Automobiles today are unsafe, and once behind the wheel, whether we are a doctor, artist, or teacher, we are all reduced to being a driver, nothing else. So we want to make life better by providing a safe way to travel in a car while being yourself.

We believe the way to achieve this goal is with autonomous vehicles. So we're working on two vehicles currently, a Level 2 and a Level 4 vehicle. In case your readers aren't familiar with autonomous vehicle terminology, there are generally four levels. With Level 1 and 2, the onus for safety is on the driver. With Level 3, 4, and 5, the liability shifts to the company which is making the vehicle.

Level 1 is most modern cars today with basic functionality like automated braking system, emergency braking, lane assistance and features like that. Level 2 is the current day Tesla which has highway autopilot, ABS, automated cruise control, lane departure warnings, etc. Past Level 2, we start moving into the realm where the safety shifts away from the driver to the car.

Level 3 is basically "eyes-off." In today's Tesla, when you drive on the freeway, you cannot take your hands off the steering wheel and you cannot take your eyes off the road. The car will help you drive, but you still need to be paying attention. But in a Level 3 vehicle, it is "Eyes-off" which means you don't have to look at the road or hold the steering wheel. However, the car can ask you to take over anytime, so you still need to sit in the driver's seat, and you cannot go to sleep in the seat.

A Level 4 is "Eyes-off. Mind-off." This means you can go to sleep and the car will be able to do a fail-safe mission completion. Even in case of a catastrophic failure, it should be able to bring you to a safe stop. Finally, Level 5 is where you don't have the steering wheel and pedals anymore, so you cannot take over the car. The car just drives itself.

As I mentioned, at NIO we are currently working on a Level 2 and a Level 4 autonomous vehicle. We launched the Level 2 at Shanghai Auto Show this year and it will be delivered early Q1 next year to our customers. This car is a seven-seat SUV called ES8. We're also working on a level 4 autonomous vehicle that we announced at South By Southwest.

"

A Level 4 is "Eyes-off. Mind-off" which means you can go to sleep and the car will be able to do a fail-safe mission completion. Even in case of a catastrophic failure, it should be able to bring you to a safe stop."

For the Level 4 vehicle, we are creating all the major technologies from the body, chassis, powertrain systems, ingress/egress systems, autonomous and fail safe systems. Because this car is "Eyes-off. Mind-off," everything has to be fail safe. That's the charter that we have. And that's what we've been working on.

Can you tell me a little bit about your role at NIO?

I've been at NIO for 14 months now. Before that I was at Tesla. I'm responsible for the autonomous vehicle backend. As you can imagine, there is a whole lot of compute in the car. A whole bunch of electronic control units which is basically the on-board computer and other compute devices. And then there is a gateway in the car which is responsible for networking the car and sending the sensor data produced by the car out to the cloud for processing.

Once data leaves through that gateway, everything that happens with the data is my responsibility. All that data is sent to what we call the Data Platform. I'm responsible for building the end-to-end experience of that platform so that autonomous driving engineers can create the algorithms, the machine learning models based off of the data to make our cars safe.

Can you talk a little bit about how NIO is using containers?

Sure, it is a very interesting story exactly how we landed where we landed today. When I started, I was coming from Telsa which as I mentioned before has a Level 2 vehicle. Since Tesla uses the cloud, I started by evaluating cloud providers, but I quickly realized that a Level 4 vehicle is a completely different ballgame. One car generates 12 to 24 terabytes of data a day in development mode. Ten cars is a whole 240 terabytes per day and there is no way I can get any of that data into clouds. So the problem statement completely changed.

Instead of focusing on data and applications, I needed to start focusing on infrastructure and systems. So I sat down to think about what's the best way to build up our infrastructure. We are a startup. We have to do it both the right way, and the cost-effective way.

At that point, we started looking at OpenStack. That architecture would be white-box hardware and OpenStack VMs running on top of hypervisors. For microservices, it was a foregone conclusion that I would use containers, and at that point, I paused. I thought, "Maybe I'm doing this the the wrong way. Why do I really need hypervisors? Why do I need the overhead?" And researching more and doing some additional proof of concepts, it felt like what we needed to build to support our massive big data use case was already really cutting edge, actually, bleeding edge of the cutting edge. So we made the choice to be container first. For one thing, we didn't have any legacy. Second I didn't want to deal with the overhead of hypervisors at all. So it made sense for me to go with a container-first strategy where I have no virtualization whatsoever.

"So today we have no virtualization. We have a bare metal server with Linux on it, the Docker daemon and containers on top of that."

So today we have no virtualization. We have a bare metal server with Linux on it, the Docker daemon and containers on top of that. The entire infrastructure is white-box based, mostly Super Micro and Lenovo. And we use orchestrators to manage the containers, both Mesos and Kubernetes. The reason being, we believe Mesos is more mature for stateful containers, and Kubernetes is very mature for stateless containers. We love the direction Kubernetes is headed, and we believe Kubernetes is getting closer to supporting stateful containers in a mature way. So we are looking out for what makes more sense for us as we move forward, but today we have both the orchestrators running in our platform.

Can you talk a little bit about some of the challenges you needed to overcome in order to run stateful services in containers on your platform?

Our use case is a pretty common Internet of Things (IoT) use case with a twist. We have a lot of data coming in from a small number of devices. This is the opposite of what you usually think about with IoT. Additionally our data is predominantly video data.

On any autonomous car you see on the road today, you see a whole bunch of cameras sitting on top of the car and around the car. There are anywhere between eight to twelve cameras that are mounted on a Level 4 car, and they're all producing high quality 1080p images and videos, which we have to ingest and process on the platform. Remember, each of these cars is generating 12-24 terabytes of data per day.

"We were happy to find a partner in Big Switch Networks whom we worked with to solve these [networking] problems. We partnered with their engineers to actually extend support for CNI interface, and we have a stable networking solution today in our platform thanks to that work."

It goes without saying that we have a whole bunch of stateful services that we have to run in order to process this data. We heavily rely on Spark, Spark ML, and TensorFlow for machine learning and deep learning scenarios, Kafka, Kafka Streaming, Hadoop, Cassandra, MongoDB, ElasticSearch, and MySQL.

We knew that while making a decision to be container first, we would need to work with partners in order to bridge some of the gaps that we have. The most critical gaps were around networking and storage.

For those interested in networking, we had challenges with overlay networking. It was almost impossible to troubleshoot issues if we had any, and IP space management was a huge challenge. With containers, we are going to have thousands of IPs which would have made it almost impossible to manage. So we really wanted an SDN network which was integrated with the CNI standards so that we can have a flatter network architecture, and we wanted to have automation of IP space management and network space management. We were happy to find a partner in Big Switch Networks whom we worked with to solve these problems. We partnered with their engineers to actually extend support for the CNI interface, and we have a stable networking solution today in our platform thanks to that work.

With networking solved, we needed to turn to storage. The challenge that we face is our data footprint is quite humongous. We will have an estimated 120 petabytes of data in a year.

"With networking solved, we needed to turn to storage. The challenge that we face is our data footprint is quite humongous. We will have an estimated 120 petabytes of data in a year."

To handle that load, we know our platform needed to provide general purpose compute and general purpose storage that could be scaled independently based on our needs. And because we're growing so fast and solving many problems for the first time, a lot of our future needs are unknown because the workloads are not there yet and we expect the workloads to evolve over time.

But for storage, we wanted a storage fabric that runs directly on our infrastructure, not alongside our infrastructure like a SAN. Like I mentioned, we are doing all white-box hardware with direct-attached storage. We needed a storage fabric that can help us create virtual volumes that integrates with both Kubernetes and Mesos in order to allow for our containers to float across multiple nodes and dish out virtual volumes as and when the containers come up and go down, clean them up, have an option to replicate these volumes.

Now part of this can be solved if you have expensive storage vendors that you use with network-attached storage, or SANs, that are available in the market. But that goes against what we are building here not to mention that a SAN for 120 petabytes wouldn't really work. We could never make it happen.

So we decided to work with vendors in this space to see how we can take existing technology and engineer it for our needs. We explored multiple options. We tried to engineer GlusterFS or Ceph for our needs and worked with another vendor in this space who had a similar product (which was not aimed towards containers) that we tried adapt. These efforts didn't work out though for a couple of reasons.

"For Kubernetes, we really didn't find a very stable way to plug in Gluster or Ceph. We started to see a lot of issues either with replication, as well as issues when failing over containers or pods across nodes."

First, GlusterFS and Ceph are not easy to operate. It takes a lot of time and effort to bring those systems up and manage them. Second, the integration with Kubernetes and Mesos is lacking. It's really tough to get that integration to work well. In this space when you are managing a system which is container-based, you're really talking about thousands of containers running to serve your workload. Because of that, you really need very tight integration between your orchestrator and storage provider.

For Gluster, we found that for Mesos there is the REX-Ray plugin available but that's very flaky and we saw a lot of performance issues, scale issues, and there was quite a bit of management overhead. Additionally, the GlusterFS and Ceph fabric themselves take quite a bit of resources off of the compute nodes running those services and didn't make sense for us.

For Kubernetes, we really didn't find a very stable way to plug in Gluster or Ceph. We started to see a lot of issues either with replication, as well as issues when failing over containers or pods across nodes.

So when those evaluations didn't work out, we finally we landed on Portworx.

For us, what we liked about Portworx was the stability, ease of use, and efficiency that was lacking in the other solutions that we evaluated. We found that the product is really built to work with Mesos and Kubernetes. It acts as a part of the Kubernetes or Mesos orchestrator which makes it really simple and easy to use.

On Mesosphere in particular, the integration is really tight. We deploy a lot Mesos Universe packages that Portworx has put together that makes it really easy for us to manage our stateful workloads, especially if you're looking at operating HDFS, Kafka, Elastic, or Cassandra. It's actually not easy to bring up and operate all these systems, especially when you're running with a very lean team, so the Portworx frameworks help a lot.

“What we liked about Portworx was the stability, ease of use, and efficiency that was lacking in the other solutions that we evaluated. We found that the product is really built to work with Mesos and Kubernetes. It acts as a part of Kubernetes or Mesos orchestrator which makes it really simple and easy to use.”

We have our first version of our platform up and serving our cars today, and I only have a team of 15 engineers out of which I have only three systems engineers. And we were able to put all of this up just because of the scheduler integration built-in and the packages Portworx built which are very intuitive and easy to use. So that’s a great feature set that I love.

Support has also been outstanding. I’ve never seen, in my entire career, this amount of support where the VP of engineering is sitting with my team trying to troubleshoot issues, which is outstanding. And a lot of this, the amount of success we had so far would not have happened without support from Portworx. So one thing I would give to them is that their focus on customer success is outstanding. It is unbelievable what they have done for us in the last few weeks.

You use a lot of open source components in your stack. Can you talk about how you think through when to use open source components and when to use a vended solution?

I believe in taking calculated risks in what we do. It is very true that we have a lot of open source products. And there is quite a bit of community support in those open source products. But there are some critical areas in your stack where you cannot afford a failure, where failure would be catastrophic and you can’t recover. In those areas, we really want to be very cautious when we make a decision on which direction to go. Storage is one key, critical, component that you don’t want to mess with. Though there are open source options, they are not as stable for our use case, so it totally makes sense in this case not to take that risk and trust a vendored solution which has been vetted out and which has been known to be very stable. Additionally, projects like CNI and CSI which create standard interfaces for cloud native applications mean that switching costs are low if that ever becomes necessary.

“There are some critical areas in your stack where you cannot afford a failure, where it is catastrophic and you can’t recover. In those areas, we really want to be very cautious when we make a decision on which direction to go. Storage is one key, critical, component that you don’t want to mess with.”

What advice would you give someone else seriously considering running stateful containers in production?

Well, first thing is make sure there is no technical debt from legacy applications that you have to deal with. If there is too much, you will struggle and you should consider other applications to containerize. Once you've decided to go container-only or container-first, think about how you are going to package the legacy into containers. There's going to be quite a bit of a leap in terms of developers understanding containers, but once they do, developing in containers is actually an amazing experience. As part of that, as I've said, you're going to need to solve your network and storage problems because those are really painful. Pick the right partners, pick the right vendors to be successful in storage and networking. Those are two key areas which I don't think there are good solutions in the open source world.

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.nio.io

@portwx

linkedin.com/company/portworx