**HashiCorp Nomad**

[HashiCorp Nomad](https://www.hashicorp.com/products/nomad) is an open-source utility that greatly reduces the complexity of automating, scheduling, and rescheduling application deployment. Nomad allows operations engineers and developers to work together more closely and improves total cost of ownership by better utilizing server hardware.

In this video, HashiCorp co-founder and CTO Armon Dadgar gives a brief, whiteboard introduction to Nomad, including:

* How to rethink the traditional deployment pattern, without changing *everything*
* Nomad does this by providing a layer between the operating system and the applications
* Nomad lets developers submit job deployment requests to Nomad, rather than making a request to operators
* Nomad automates operational tasks — automatic rescheduling, transparently removing live services from a node, etc. — without needing to coordinate with developers
* Nomad typically lets you gain a 10x improvement in utilization
* Nomad schedules not just containers, but also apps that can't easily be containerized
* Nomad also helps you manage a fast-moving, rate-limited queue of jobs — e.g., an event-triggered or serverless pattern
* Nomad simplifies high-performance computing — e.g., scheduling 40 million containers in minutes

**Transcript**

Hi. My name is Armon Dadgar and today I wanted to do a brief introduction to Nomad. So, when we talk about Nomad one of the things, we see that's really common is this deployment pattern of a single operating system with a single application running on top of it. And in this configuration, they meet on top of a single VM.

Now the challenge that we often see with this configuration is: you really have two distinct audiences here.

* You have your developer audience who cares about the application lifecycle. They care about scaling up, scaling down, changing configuration, deploying their new version.
* But at the same time, you have your operator audience, and they care about a different set of things. So, when we talk about our operators, they care about “Are we running the right version of the OS, is it patched, do we have enough capacity in our fleet?”.

And the challenge is: Although they have an independent set of concerns, they have to coordinate, because the application is running on an OS, on a VM, at the end of the day.

**DevOps mediated**

So, what we often see is the Development group has to file a ticket. So anytime the development group wants to do anything application lifecycle related, they have to file a ticket against the Operations group. And that's the layer at which the coordination is done. So, the first thing we're really looking at when we talk about Nomad is: How do we split this so that we can have independent workflows?

The primary goal of Nomad is to sit in between here and mediate --really provide a layer where we have a southbound API focused on the operator and a northbound API focused on the developer. So, what does this really mean?

For the developer what we want to do is let them write their job. This is what Nomad calls it: it's a job file, which is an infrastructure's code way of declaring everything about their job. So, it would say I have this web application I want to run. It's version 10 of my application and I want three instances of it running.

And now the developer just submits this job file to Nomad's API and it's Nomad's responsibility to find space in this cluster to run three instances of this web server.

So, we might have a 100-node cluster, and Nomad's going to find three machines that have available capacity and deploy the web server there. And now, as a developer when we come back and say, “You know what, I want to deploy version 11 of my application,” I simply change my job file and then specify what rollout strategy do I want.

Do I want to use a canary? Do I want to do a blue/green deployment? Do I want to do a rolling deploy? So, I can specify my strategy for deploying my application and then I submit it to Nomad and Nomad takes care of rolling out this change across the fleet safely.

**Automation**

In addition to just managing deploying our application as well as making changes to it as we change version or scale up and down. So, we could easily come in here and just change three to five, and Nomad would go and run two more copies of this... The other thing is: how do we automate some of the operational challenges that are historically part of the Operations group?

And so, when we talk about these sorts of operational issues, it's things like how do we make sure, if this application crashes, that it gets restarted. So, we want to make sure we can gracefully restart the application and ensure it stays online, even though it might have crashed or experienced an issue.

The other side of it is: what if the machine that we're running on, or the rack, or the cage, or the data center that we're running on, fails? What we really want to do is reschedule that somewhere else. So, if Nomad detects that the machine running the application has failed, it will find somewhere else with available capacity to run it.

And so, these are traditionally things where we may have paged someone to go deal with this operational issue and provide a reliable service, but instead Nomad lets us automate it.

Now that's the developer focus in this sort of life cycle. As we talk about the operator, they really focus on the southbound side and they have a few different concerns which is, like I said: are there enough machines in the fleet with capacity? Are they patched? Are we running the latest version?

So, they have a set of needs as well. They need to be able to come and say, "You know what, these 10 machines, I want to take them out of the fleet so that I can patch them and then bring them back". And so, they have an API as well where they can come and say, "You know what, I'd like to gracefully drain these 10 machines and over the next four hours get all the workload off of them and then I can take it out of service, do a patch, bring it in and re-allow the workload to go on it".

So, this is how we think about these two different audiences. What does the developer need for app lifecycle, and how do we let them define their job requirements in the “infrastructure as code” way? And from an operations side, how do we decouple them and allow them to do the things they care about - cluster management and load management - without tightly coordinating with developers. So, this is kind of the first-level goal.

**Boosting TCO**

Now the second-level challenge we see with Nomad is: when you look at most infrastructure, you have a really bad rate of hardware utilization - typically less than 2%. So how do we actually solve this? Because we have all these eight-core, sixteen-core machines, that are running an application doing a hundred or a thousand requests a day - where this is effectively idle.

We're not making good use of hardware. And so, the approach Nomad takes is to run multiple applications on the same machine.

And so how do we move from a place where we're at less than 2% to being at 20% to 30% utilization? Now you might look at this and say 20% to 30% utilization doesn't sound that good, right? Why don't we shoot even higher than that? But what we have to realize is the law of small numbers. Because we're starting at such a bad place, going from 2% to call it 20%, what you still get out of this transition is an incredible reduction in your fleet size. So, as we go from 2% to 20% it’s actually a 90% reduction in the amount of overall hardware we need. We can replace every ten machines with one, basically.

So, there's this great total-cost-of-ownership optimization that comes from running multiple applications and making better use of our resources. So, this is the kind of primary focus. How do we allow this for decoupling and self-service? How do we look at total cost of ownership as a secondary goal?

**Containers and legacy apps**

And so then, what we haven't really mentioned is, here we're talking generically about an application running on a machine. This really comes back to how flexible Nomad is.

So, on one side, a major use-case for Nomad, is acting as a container platform. So, this application that we're deploying might be packaged as a Docker container. Specify as part of our job file that our web server is using this container, let's say web-V10, then we hand that to Nomad to do the deploy.

But what about applications that aren't containerized or can't easily be containerized. This is actually a whole 2nd use-case for Nomad. Which is both Windows as well as legacy applications.

So, when we talk about some of these applications, maybe it's just a simple C# application that we're deploying on Windows or it's something more heavyweight that we can't easily containerize. Nomad allows us to run many of these types of workloads without needing to make that sort of transition and packing format. So, a common use-case is running C# apps directly on top of Windows without containerizing or porting them to Linux. So, this ends up being a common workflow for us.

**Scheduling**

Now beyond that, the interesting thing is when we talk about this north- and southbound API, what we're really providing is an API for scheduling work. So, it could be that we are specifying our job in the form of this job file and some talk and providing that and submitting it manually to Nomad. But we could also programmatically consume Nomad's API to deploy a job. And so, this actually leads to a few interesting use cases. One of these we might call it job-queue, serverless pattern of deployment.

When an event comes in, how do we translate that event into something that needs to execute. A great example of this is CircleCI. Every time a commit comes in, CircleCI has to trigger a build that goes and tests: does this cause a change, yes or no? CircleCI has publicly talked about how they use Nomad behind their systems for their infrastructure. They get a web hook in an event that a commit has taken place, they translate that and submit a job to Nomad. Now go run that build. And what they see is being able to submit well over a thousand jobs a minute to Nomad.

And so, in this sense it's acting in two ways. We're queuing up jobs for Nomad to run. It might be that for a temporary period of time the number of incoming events exceeds our ability to process it. So, our rate in might be a thousand a minute but we only have enough hardware capacity to process 500 or 800 events a minute. Nomad will allow this work to back up, and queue it until there's available capacity, and drain it as it comes in. This allows us to also start to think about the serverless paradigm. Right?

How do I think about transforming an event and turning that into a small unit of work that just processes that event and scheduling all of this work independently? So, this becomes an interesting use case, because we have this API that we can use to programmatically consume infrastructure. Now one of the interesting things when we talk about CircleCI's use case, we're talking about a relatively large scale. Right? A thousand different events per minute.

But this is actually relatively easy-going for Nomad. Because a whole other use case for us is high performance computing.

So, we talk about high performance computing, there's many different kinds of interesting use-cases here, right? It could be that you're a financial institution that every night you want to run a complex risk model. So, you’re sending up 100,000 cores, running these complex risk calculations to determine: should I buy or sell stock? Or am I overexposed in certain areas? Where what really matters to you is being able to consume an enormous amount of compute for a period of time and really caring about how long does it take you to compute some job or some calculation?

And so, this is an interesting use case that we benchmarked very publicly in what we called the C1N or our million-container challenge. Which we looked at: how quickly could we schedule a million containers on a cluster of 5,000 machines? And what we found was we could actually run all million containers, which were an instance of Redis in less than five minutes.

So, that's an incredible rate of scheduling - and at the time, we thought this sets an upper bound of what's reasonable and what we'd actually see customers doing. But what we found in practice, and Citadel has publicly talked about it - for those unfamiliar, Citadel is a large hedge fund - they spoke to us and said, "This is cute, but could you actually do this at 40 times this scale?" And so, their use case is very much like the one I described, where periodically they want to run these incredibly large calculations and simulations where speed is of the essence.

It affects their ability to make a trade within the same day until this calculation is completed. And so, they want to be able to scale to incredibly large clusters with thousands and thousands of cores, quickly run these massive-scale computations, and then spin this cluster back down. And so, the ability to programmatically generate and submit these jobs and then queue them up -- so they might be running multiple of these things on a fixed set of hardware -- is a powerful feature of Nomad, above and beyond this sort of self-service infrastructure capability.

**Summary**

Again, when we talk about Nomad, it's really starting to look at how do we move away from this tight coupling of the application from the operating system and introduce a layer of abstraction. This layer of abstraction buys us both a north- and southbound API for cluster management, as well as application management.

A secondary side-effect of that is doing automatic cost optimization by bin packing -- placing multiple application on the same machine -- and this really enables us four distinct types of use-cases and patterns that we see around Nomad.

Hopefully this was a useful high-level introduction to Nomad. There's a lot more material available on our website, as well as content that goes a lot deeper than this. So, I encourage you to check it out. Thank you so much.