Kubernetes Vs. AWS ECS Vs. Docker Swarm Vs. Serverless

[](https://cache.logicworks.com/wp-content/uploads/2018/02/12103037/iStock-629285904.jpg)

Docker orchestration tools and serverless are getting a lot of hype — but what’s the best tool for your application, and how do you actually deploy them on AWS?

Last week, our Sr. Solutions Architect and DevOps Engineer Phil Christensen put on a fantastic webinar comparing containers and serverless on AWS. Below is a recap of the webinar. To watch the full replay, go here or watch below.

**Highlights of the Webinar:**

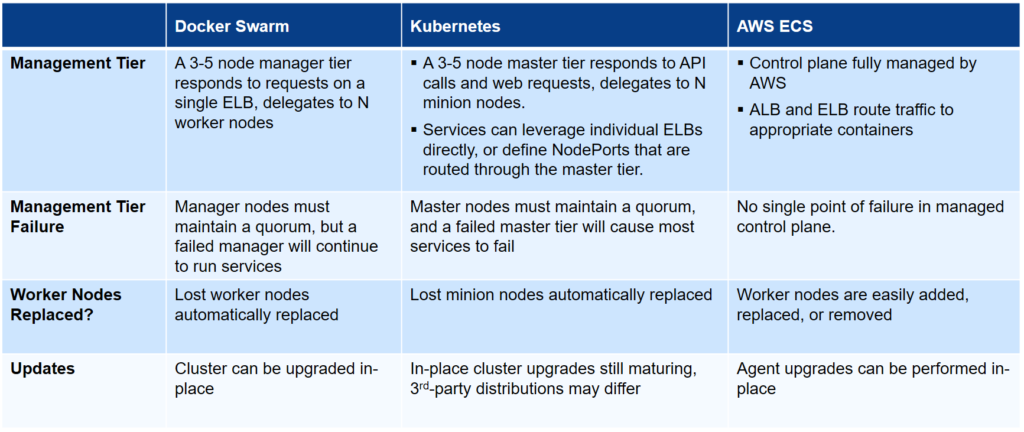
***Intro to Containers***

* **The standard deployment model** these days for any kind of application that you deploy on the cloud is changing very rapidly. But the process of deploying an application is usually task intensive. There are many challenges due to the needs of running various applications and different frameworks and architectures on the same set of computing resources.
* The two options that have been in recent months become particularly prominent in this space are **Containers**and what people are calling **Serverless**.
* Our position is that these are two technologies that are **essential for anybody working in the space** to understand. Because depending on the task that you’re looking at, depending on your project at hand, one or the other may be a better choice for your particular organization.
* The first benefit of a containerized development approach is the fact that you can now **deploy the exact same artifact to your server that was created in development**.
* We’re gonna be running that same set of assets and the only thing that we **change is some of the configuration variables** and so forth that are presented to a Docker containers as environment variables, mounted volumes, memory and CPU limitations.

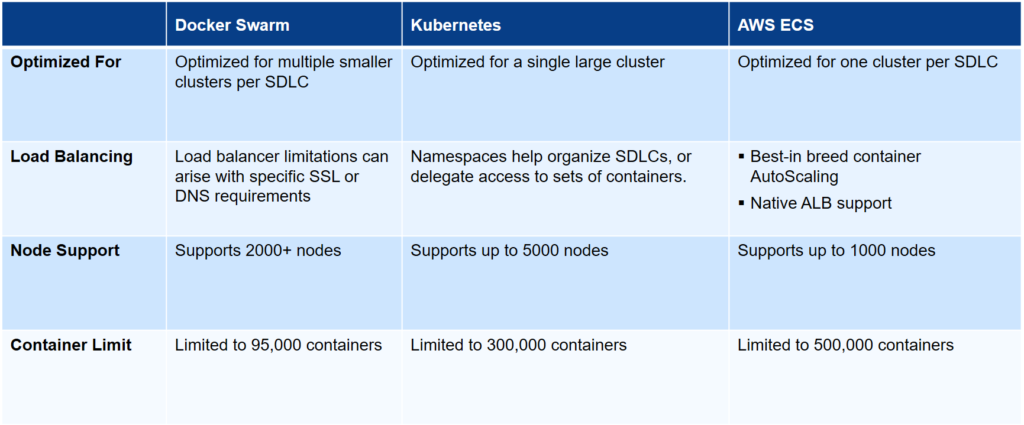
***Container Orchestration***

* To get the best benefits out of the Docker containers in your architecture, you need some software to be responsible for moving the containers around to respond to things like auto scaling events, changes and availability if there’s a failure of an application node or if there’s a failure of the backing host that actually runs that container. This is container orchestration.
* **Docker Swarm**: It’s oftentimes very high on people’s list of things to investigate because it’s got that Docker name brand recognition. It does ship directly with Dockers. So if you’ve installed Docker on any of your servers, you also have the necessary tools installed to run Docker Swarm. It has one of the simplest configurations of any of the orchestration tools. But it does have some limitations and some things that you should keep in mind if you’re thinking to leverage it.
  + First of all, it has somewhat **limited cloud integration**. It does integrate with things like AWS load balancers and other load balancing tools. And it has some limited service discovery capabilities. But at the end of the day, it is not particularly aware of a cloud environment that its running in.
  + Docker Swarm tends to be particularly useful for people who are trying to get comfortable with an orchestrated environment or **who need to adhere to a simple deployment technique** but also have more just one cloud environment or one particular platform to run this on.
  + But again, Docker Swarm is one of those things that anybody who has Docker has a copy of this, and it’s something that’s worth considering as a orchestration option.
* **Kubernetes**: This is kind of the elephant in the room in a lot of ways. Kubernetes itself is probably the most feature filled orchestration layer. It has an extremely large community supporting it but across many different platforms. Whether it’s on OpenStack, AWS, Azure, it has extremely powerful built-in container discovery and this really comes into play for many of the different ways that it works.
  + Kubernetes is a great way to handle a **very large amount of undifferentiated compute**. It’s good in the data center and it also is increasingly very useful in the AWS cloud especially for some of its support for the native AWS cloud services. So if you do a particular Kubernetes deployment, it will be aware of some of the resources that need to stand up for you.
  + Kubernetes right now has a **fairly complex install and configuration process**. And it can be somewhat difficult to stay up to date because it’s constantly coming out with new revisions of the Kubernetes software.
* **AWS Elastic Container Service**: Currently we’re talking about this separate from the Elastic Kubernetes Service that’s currently in preview. One of the good things about ECS is that it tends to focus on just the things that you really need to have to have a container deployment, and the fact that there are endless other services that are suitable replacements for some of the discovery options and so forth that are available in Kubernetes.
  + Additionally, because of the fact of an entirely AWS managed schedule or tier or management tier, we can really be a lot more confident about the overall reliability of the platform.
  + ECS is probably my most commonly recommended orchestration layer for people on the AWS cloud.

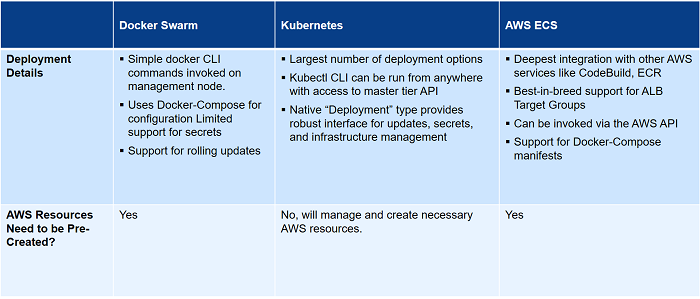
***Achieving High Availability***



***Cluster Capacity***



***Deployment Details***



***Serverless***

* When you build something with serverless, you’re going to **exchange flexibility for scale**. When we write to a serverless API, whether we use Amazon SAM Local toolkit, the [serverless.com](https://serverless.com/) toolkit which is really excellent, we’re going to leverage the existing frameworks.
* For an AWS serverless deployment, oftentimes we start with a Lambda function, and deploying that Lambda function is a matter of taking a set of code and dropping it in S3 and telling Lambda where that code is. Then we have a number of events that we need to send to those Lambda functions to trigger them to execute.
* The creation of the API gateway and related services and other platforms allows us to define a REST API that we’ll then invoke this function in a certain manner.
* If we go as far as using something like DynamoDB, **we can create extremely complex applications** (both front facing and back office facing) using entirely Lambda functions and a few other tools to link the functions together.
* So why Serverless? We’ve got **very quick startup times**. We don’t have separation between different clusters. It’s a entirely undifferentiated pool and there’s **no need for orchestration** because AWS or the cloud providers are going to handle it for us.
* **We only pay for what we use**. And then because of that, we have the ability to deploy an arbitrary number of development instances or software development lifecycle environments at little or no cost. If our developed environments are only used by a team of 5-10 people, we’re going to see substantially less charges on that than we do on our production site.
* You have a little bit less flexibility around development techniques particularly around things like very long running tasks. All of the function services right now have to have a governor that ensures you don’t exceed a certain amount of run time, a certain amount of CPU or memory usage because that’s the way to make a shared computing resource really work.
* There’s always going to be an upper limit of the amount of time or the amount of memory that a given process is allowed to use, so we can leverage technology like **AWS Step Function** to take a series of Lambda function and orchestrate them. This is to make sure that a particular state machine sends a certain number of arguments, parameters, etc. to the beginning of your pipeline and it will follow that chain of Lambda functions and continue to modify that state until you get to the end of your function and your task is complete.
* This can be challenging; it takes some time to r**earrange your thinking** to effectively use an asynchronous workflow style of development.
* **Now what are some good use cases for Serverless?**
  + For the longest time in AWS, we would always use SNS to send communications from one place to another. But at a certain point you have to have something reading a queue and responding to those messages and usually that meant an EC2 instance or something to that effect. Now, our SNS topics can directly invoke a Lambda function.
  + We can write API-only applications.
  + With some of the tools available such as the Zappa toolkit for integrating with Django and some of the other toolkits, you have the opportunity to do any kind of web facing application by taking a series of Lambda functions and integrating them with the API Gateway Service. Keep in mind that a REST API is really no different than a webpage so that what if the webpage happens to return HTML, that’s fine too.
  + It’s also useful for really long running or deeply integrated AWS scripts or applications.

Want more information on serverless and containers? Phil goes into even more detail in the webinar, [which you can view here](http://go.logicworks.com/webinar-kubernetes-webinar-replay-201801-0).

[**Logicworks**](http://www.logicworks.com/)**helps customers plan, build, and run software defined infrastructure. Logicworks has helped hundreds of clients including Major League Soccer and Makerbot deploy a**[**DevOps**](http://www.logicworks.com/devops-automation/)**driven approach to enable greater availability, security, and performance. For more information, please contact info@logicworks.com or (212) 625-5300.**