What Is Serverless?

What is Serverless?

Serverless techniques and technologies can be group into two areas:

- Backend as a Service (BaaS) Replacing server-side, self-managed components with off-the-shelf services
- Functions as a Service (FaaS) A new way of building and deploying server-side software, oriented around deploying individual functions

The key is that with both, you don't have to manage your own server hosts or server processes and can focus on business value!

Serverless doesn’t mean there are no servers, it means you don’t care about them.

What is Serverless?

Key characteristics

A Serverless service …

- … does not require managing a long-lived host or application instance
- … self auto-scales and auto-provisions, dependent on load
- … has implicit high availability
- … has performance capabilities defined in terms other than host size/count
- … has costs that are based on precise usage, up from and down to zero usage

Why Serverless?

- Shorter lead time
  - Reduced packaging and deployment complexity
- Increased flexibility of scaling
- Reduced resource cost
- Reduced labor cost
- Reduced risk

Drawbacks / Limitations of Serverless

Part 1/2

- Unpredictable costs
- Spinning up machines takes time - from a few seconds to minutes
- Most Serverless applications are stateless and the management of state can be somewhat tricky
- Higher latency due to inter-component communication over HTTP APIs and “cold starts”
- Problematic with downstream systems that cannot increase their capacity quickly enough
- Typically limited in how long each invocation is allowed to run
- Multitenancy problems

Drawbacks / Limitations of Serverless

Part 2/2

- Debugging is more complicated (a single request can travel between several machines and some of those machines disappear at times)
- Added work is needed to provide tracing and monitoring solutions, which can add complexity and cost to the project
- Security can be more demanding in a serverless environment
- Loss of control over
  - absolute configuration
  - the performance of Serverless components
  - issue resolution
  - security
- The difficulty of local testing
- Vendor lock-in unless you are using OSS projects like e.g. Knative

What is Cloud Native Runtimes for VMware Tanzu

Serverless runtime that supports both app & function programming models on K8s

Foundational components:

- Knative Serving
- Knative Eventing
- AWS Sources
- Streaming Runtime
- Batch Runtime
- vSphere Sources

Optimized Tanzu integrations:

- Tanzu Observability / Wavefront
- Tanzu Service Mesh / AVI LB / Contour
- Tanzu Build Service – Application & Function Buildpacks

Runs on Kubernetes:

- TKG & TKGi
- Public Cloud - EKS, GKE, AKS
- Openshift, Rancher and Local K8s Tools

Deployment model:

- Leverages Carvel tools
- Available on TanzuNet
- Installable via TMC

What is Knative?

Knative is an open-source community project which adds components for deploying, running, and managing applications on any Kubernetes in a Serverless way.

Unlike earlier serverless frameworks, Knative was designed to deploy any modern app workload - everything from monolithic apps to microservices and tiny functions

It consists of two primary components:

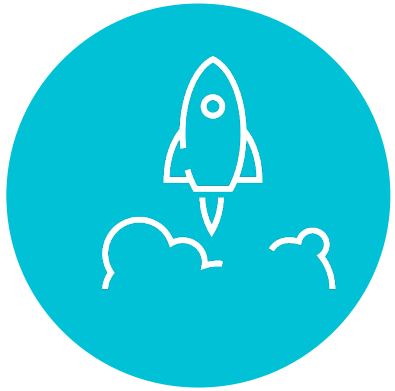
- Serving: Provides middleware components that enable rapid deployment, upgrading, routing, and automatic scaling of containers through a request-driven model for serving workloads based on demand
- Eventing: A system for consuming and producing events to stimulate apps. Apps can be triggered by a variety of sources, such as events from your own apps or cloud services from multiple providers

Benefits of Cloud Native Runtimes for VMware Tanzu

Reduce Excess Capacity

- Scale-to-zero capabilities
- Pay only for used resources
- Reduce public cloud costs
- Optimize on-premise server usage

Quickstart Kubernetes Development



- Abstract Kubernetes concepts without hiding them
- Allow developers to quickly & easily begin developing on Kubernetes

Advanced Deployment Capabilities

- Roll out new versions of apps easily with high confidence
- In-built traffic splitting
- Easily follow Blue/Green or Canary deployment patterns

App & Function Programming Model

- vSphere, AWS, Kafka, RabbitMQ event sources
- Function Buildpacks – Java, .NET, NodeJS, Python
- HTTP, Cloud Event, Batch & Streaming triggered functions

No Vendor Lock-in

- Build on OSS frameworks like Knative, Buildpacks etc.
- Same experience across multiple clouds
- Robust OSS community supporting the project