Containerised ASP.NET Core apps with Docker and Kubernetes

Mete Atamel
Developer Advocate @ Google
Mete Atamel
Developer Advocate for Google Cloud
@meteatamel
atamel@google.com
meteatamel.wordpress.com

Please send talk feedback: bit.ly/atamel
The .NET Revolution
Things are changing

- **2014**: .NET Goes Open Source
- **2015**: Ubuntu, Debian on Azure
- **2016**: PowerShell on Linux
- **2017**: OpenSSH on Windows
- **2018**: SQL Server on Linux
- **2019**: Bash on Windows
- **2020**: Microsoft joins Linux foundation
- **2021**: ASP.NET Core on Linux, Mac, Windows
- **2022**: Google joins .NET foundation

@meteatamel
The convergence

- SQL Server
- Eclipse
- Bash
- Java
- PowerShell
- ASP.NET
- Apache
- Visual Studio
- C#
- MySQL
Great time to be a .NET developer!
However, software development is HARD!
And it is not getting any easier...
In the good old days

This is all I had to care...

Life was good :-(
A lot happened since then

Internet

- Databases
- App Servers
- Firewalls
- Object Oriented Programming
- DevOps
- Web Servers
- Microservices
- Version Control
- Virtual Machines
- The Monolith
- Mobile
- Cloud Computing
- Caching
- Machine Learning
- Big Data
- IoT

Google Cloud Platform
Nowadays
We haven’t even talked about

Maintaining code in **different** languages on **different** types of machines

Rolling out the **new version** of your code reliably

Rolling back to the **old version** if something goes wrong

Managing **configuration** and **secrets**

Managing **scripts** that need to run on each machine
I just want to write some code to solve a real-world problem
The reality

This is all I **want** to care

- Load Balancer
- Health Endpoint
- Health Endpoint
- Health Endpoint
- Health Endpoint
- Health Checker

This is all I **have** to care

- Load Balancer
- Task Queues
- Health Endpoint
- Health Endpoint
- Health Endpoint
- Health Endpoint
- Health Checker

ZONE A

ZONE B

REPLICATION

Autoscaler

This is all I **want** to care

This is all I **have** to care
Writing code to solve problems is still fun!

Running that code in production is very hard

What do we do?
In the good old days

Write your code, pass it to QA for testing, let operations team run it...

It's not my problem!
Nowadays, it is your problem

What do we do?
You can write your code in any language and run anywhere exactly the same way.

Your app is optimally deployed somewhere and managed by someone. It just works!

There are no machines. All resources are automatically provisioned on demand.
Docker + Kubernetes + Cloud

Write your code in any language and run it anywhere exactly the same way
⇒ Containers (eg. Docker, Rkt)

Your app is optimally deployed and managed
⇒ Container Management Platforms (eg. Kubernetes, Docker Swarm, Mesos)

All the resources needed for your app is automatically provisioned per demand
⇒ Cloud Providers (eg. Google Cloud, AWS, Azure)
Demo: Simple Microservice
Containers
What is a container?

A **lightweight** way to virtualize applications

Linux (or Windows) processes

<table>
<thead>
<tr>
<th>Lightweight</th>
<th>Easily deployable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hermetically sealed</td>
<td>Introspectable</td>
</tr>
<tr>
<td>Isolated</td>
<td>Composable</td>
</tr>
</tbody>
</table>
Why containers?

Physical Machine
- ✗ No isolation
- ✗ Common libs
- ✗ Highly coupled Apps & OS

Virtual Machines
- ✓ Isolation
- ✓ No Common Libs
- ✗ Expensive and Inefficient
- ✗ Hard to manage

Containers
- ✓ Isolation
- ✓ No Common Libs
- ✓ Less overhead
- ✗ Less Dependency on Host OS
Google has been developing and using containers to manage our applications for over 12 years.
Everything at Google runs in containers

Gmail, Web Search, Maps, ...
MapReduce, batch, ...
GFS, Colossus, ...
Even Google’s Cloud Platform: our VMs run in containers!

We launch over 2 billion containers per week
Demo: Containerised Microservice
Containers not enough

Containers help to create a lightweight and consistent environment for apps

But you still need to answer these questions:

- Who takes care of redundacy?
- Who takes care of resiliency?
- Who scales up/down your app?
- Who and how a new version of your app gets deployed?
- Who rolls back to a previous version if something goes wrong?
- Etc. etc. etc.
Kubernetes
Greek for "Helmsman"; also the root of the words "governor" and "cybernetic"

- Manages container clusters
- Inspired and informed by Google’s experiences and internal systems (borg)
- Supports multiple cloud and bare-metal environments
- Supports multiple container runtimes
- **100% Open source**, written in Go

Manage **applications**, not machines
The 10000 foot view

users

master

nodes

API

CLI

UI

apiserver

etcd

scheduler

controllers

kubelet

kubelet

kubelet

kubelet

@meteatamel
All you really care about

@meteatamel
Container clusters: A story in two parts

1. Setting up the cluster
   • Choose a cloud: GCE, AWS, Azure, Rackspace, on-premises, ...
   • Choose a node OS: CoreOS, Atomic, RHEL, Debian, CentOS, Ubuntu, ...
   • Provision machines: Boot VMs, install and run kube components, ...
   • Configure networking: IP ranges for Pods, Services, SDN, ...
   • Start cluster services: DNS, logging, monitoring, ...
   • Manage nodes: kernel upgrades, OS updates, hardware failures...

Not the easy or fun part, but unavoidable

This is where things like Google Container Engine (GKE) really help
Kubernetes cluster on GKE
Demo: Create Kubernetes cluster
2. **Using the cluster**
   - Run Pods & Containers
   - Replica Sets
   - Services
   - Volumes

This is the fun part!

A distinct set of problems from cluster setup and management

Don’t make developers deal with cluster administration!

Accelerate development by focusing on the applications, not the cluster
Kubernetes Building Blocks
<table>
<thead>
<tr>
<th>Kubernetes Terminology</th>
<th>Deployment</th>
<th>ReplicaSet</th>
<th>DaemonSet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pod</td>
<td>Liveness Probe</td>
<td>Job</td>
</tr>
<tr>
<td></td>
<td>Volume</td>
<td>Readiness Probe</td>
<td>Stateless</td>
</tr>
<tr>
<td></td>
<td>Label</td>
<td>Service</td>
<td>ConfigMap</td>
</tr>
<tr>
<td></td>
<td>Selector</td>
<td></td>
<td>Secret</td>
</tr>
</tbody>
</table>
Container cluster

Each pod contains one or more containers

Service communication channel

Replication controller
Replicas: 3

Role: frontend
Env: prod

Blueprint
“pod template”

Nodes

Pods

Replication controller
Replicas: 3

Role: frontend
Env: prod

microservice

registry
containers
Deployments
A Deployment provides declarative updates for Pods and Replica Sets

Describe the desired state and the Deployment controller will change the actual state to the desired state at a controlled rate for you.

Deployment manages replica changes for you
• stable object name
• updates are configurable, done server-side
• kubectl edit or kubectl apply
Demo: Create Deployment
# Kubernetes Terminology

<table>
<thead>
<tr>
<th>Kubernetes Terminology</th>
<th>Equivalent in Kubernetes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Deployment</strong></td>
<td></td>
</tr>
<tr>
<td>Pod</td>
<td>ReplicaSet</td>
</tr>
<tr>
<td>Volume</td>
<td>Liveness Probe</td>
</tr>
<tr>
<td>Label</td>
<td>Readiness Probe</td>
</tr>
<tr>
<td>Selector</td>
<td>Service</td>
</tr>
<tr>
<td><strong>DaemonSet</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ReplicaSet</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Job</strong></td>
<td></td>
</tr>
<tr>
<td><strong>StatefulSet</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ConfigMap</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Secret</strong></td>
<td></td>
</tr>
</tbody>
</table>
Pods and Volumes
**Small group** of containers & volumes

**Tightly** coupled

The atom of scheduling & placement

Shared namespace
- share IP address & localhost
- share IPC, etc.

Managed lifecycle
- bound to a node, restart in place
- can die, cannot be reborn with same ID

**Example:** data puller & web server
Pod-scoped storage

Support many types of volume plugins

- Empty dir (and tmpfs)
- Host path
- Git repository
- GCE Persistent Disk
- AWS Elastic Block Store
- Azure File Storage
- iSCSI
- Flocker
- NFS
- vSphere
- GlusterFS
- Ceph File and RBD
- Cinder
- FibreChannel
- Secret, ConfigMap, DownwardAPI
- Flex (exec a binary)
- ...

Volumes
# Kubernetes Terminology

<table>
<thead>
<tr>
<th>Deployment</th>
<th>ReplicaSet</th>
<th>DaemonSet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pod</td>
<td>Liveness Probe</td>
<td>Job</td>
</tr>
<tr>
<td>Volume</td>
<td>Readiness Probe</td>
<td>StatefulSet</td>
</tr>
<tr>
<td>Label</td>
<td>Service</td>
<td>ConfigMap</td>
</tr>
<tr>
<td>Selector</td>
<td></td>
<td>Secret</td>
</tr>
</tbody>
</table>
Labels & Selectors
Arbitrary metadata
Attached to any API object
Generally represent **identity**
Queryable by **selectors**
  • think SQL ‘select ... where ...’
The only grouping mechanism
  • pods under a ReplicationController
  • pods in a Service
  • capabilities of a node (constraints)
App = MyApp
App = MyApp, Role = FE
App = MyApp, Role = BE
App = MyApp, Phase = prod
App = MyApp, Phase = test
# Kubernetes Terminology

<table>
<thead>
<tr>
<th>Deployment</th>
<th>ReplicaSet</th>
<th>DaemonSet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pod</td>
<td>Liveness Probe</td>
<td>Job</td>
</tr>
<tr>
<td>Volume</td>
<td>Readiness Probe</td>
<td>StatefulSet</td>
</tr>
<tr>
<td>Label</td>
<td>Service</td>
<td>ConfigMap</td>
</tr>
<tr>
<td>Selector</td>
<td></td>
<td>Secret</td>
</tr>
</tbody>
</table>
Resiliency & Redundancy
ReplicaSets*

A simple control loop
Runs out-of-process wrt API server

**One job**: ensure N copies of a pod
- grouped by a selector
- too few? start some
- too many? kill some

Layered on top of the public Pod API

Replicated pods are **fungible**
- No implied order or identity

* The evolution of ReplicationControllers
ReplicaSets

ReplicaSet

#pods = 3
app = demo
color in (blue, grey)

Pod

app = demo
color = blue

Pod

app = demo
color = blue

Pod

app = demo
color = grey

Behavior

- Keeps Pods running
- Gives direct control of Pod #s
- Grouped by Label Selector

Benefits

- Recreates Pods, maintains desired state
- Fine-grained control for scaling
- Standard grouping semantics
Demo: ReplicaSets
Kubernetes Health Checks
It’s **your** responsibility to let Kubernetes know whether your app is healthy or not!
Liveness Probes

Liveness Probes make sure your application is running

livenessProbe:
  # an http probe
  httpGet:
    path: /healthz
    port: 8080
  initialDelaySeconds: 15  # wait 15 seconds after pod is started to check for health
  timeoutSeconds: 1       # wait 1 second for a response to health check
Readiness Probes

Readiness probes make sure your application is ready to serve traffic

readinessProbe:
  # an http probe
  httpGet:
    path: /readiness
    port: 8080
  initialDelaySeconds: 20  # wait 20 seconds after pod is started to check for health
  timeoutSeconds: 5  # wait 5 second for a response to health check
<table>
<thead>
<tr>
<th>名词</th>
<th>Deployment</th>
<th>ReplicaSet</th>
<th>DaemonSet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pod</td>
<td></td>
<td>Liveness Probe</td>
<td>Job</td>
</tr>
<tr>
<td>Volume</td>
<td></td>
<td>Readiness Probe</td>
<td>StatefulSet</td>
</tr>
<tr>
<td>Label</td>
<td></td>
<td>Service</td>
<td>ConfigMap</td>
</tr>
<tr>
<td>Selector</td>
<td></td>
<td></td>
<td>Secret</td>
</tr>
</tbody>
</table>
Services
A logical grouping of pods that perform the same function (the Service’s endpoints)

- grouped by label selector

Load balances incoming requests across constituent pods

Choice of pod is random but supports session affinity (ClientIP)

Gets a **stable** virtual IP and port

- also a DNS name
Demo: Services
Kubernetes Dashboard

A general purpose, web-based UI to view/manage Kubernetes clusters
Demo: Kubernetes Dashboard
# Kubernetes Terminology

<table>
<thead>
<tr>
<th>Deployment</th>
<th>ReplicaSet</th>
<th>DaemonSet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pod</td>
<td>Liveness Probe</td>
<td>Job</td>
</tr>
<tr>
<td>Volume</td>
<td>Readiness Probe</td>
<td>StatefulSet</td>
</tr>
<tr>
<td>Label</td>
<td>Service</td>
<td>ConfigMap</td>
</tr>
<tr>
<td>Selector</td>
<td></td>
<td>Secret</td>
</tr>
</tbody>
</table>
Scaling
ReplicaSet
version = v1
type = FE
#pods = 3

Service
name = frontend
Label selector:
type = BE
Service
name = frontend
Label selector:
type = BE
Demo: Scaling
Rolling Update
ReplicaSet
- name: my-app-v1
- replicas: 3
- selector:
  - app: MyApp
  - version: v1
ReplicaSet
- name: my-app-v1
- replicas: 3
- selector:
  - app: MyApp
  - version: v1

Service
- app: MyApp

ReplicaSet
- name: my-app-v2
- replicas: 0
- selector:
  - app: MyApp
  - version: v2
ReplicaSet
- name: my-app-v1
- replicas: 3
- selector:
  - app: MyApp
  - version: v1

Service
- app: MyApp

ReplicaSet
- name: my-app-v2
- replicas: 1
- selector:
  - app: MyApp
  - version: v2
ReplicaSet
  - name: my-app-v1
  - replicas: 2
  - selector:
    - app: MyApp
    - version: v1

ReplicaSet
  - name: my-app-v2
  - replicas: 1
  - selector:
    - app: MyApp
    - version: v2

Service
  - app: MyApp
Service
- app: MyApp

ReplicaSet
- name: my-app-v1
- replicas: 2
- selector:
  - app: MyApp
  - version: v1

ReplicaSet
- name: my-app-v2
- replicas: 2
- selector:
  - app: MyApp
  - version: v2
ReplicaSet
- name: my-app-v1
- replicas: 1
- selector:
  - app: MyApp
  - version: v1

Service
- app: MyApp

ReplicaSet
- name: my-app-v2
- replicas: 2
- selector:
  - app: MyApp
  - version: v2

Rolling Update
ReplicaSet
- name: my-app-v1
- replicas: 0
- selector:
  - app: MyApp
  - version: v1

Service
- app: MyApp

ReplicaSet
- name: my-app-v2
- replicas: 3
- selector:
  - app: MyApp
  - version: v2

Rolling Update

Google Cloud Platform
Rolling Update

Service
- app: MyApp

ReplicaSet
- name: my-app-v2
- replicas: 3
- selector:
  - app: MyApp
  - version: v2
Demo: Rolling Update
Canary Deployments

ReplicaSet
version = v1
type = BE
#pods = 2

Pod
version = v1
type = BE

Pod
version = v1
type = BE

Pod
version = v2
type = BE

Service
name = backend
Label selector:
type = BE

ReplicaSet
version = v2
type = BE
#pods = 1
ReplicaSet
name=locust
role=worker
#pods = 4

Scale
CPU Target% = 50

Heapster

70% CPU

40% CPU

> 50% CPU

Pod
name=locust
role=worker

Pod
name=locust
role=worker

Pod
name=locust
role=worker

Pod
name=locust
role=worker

@meteatamel
DaemonSets
Problem: how to run a Pod on every node?
  • or a subset of nodes

Similar to ReplicaSet
  • principle: do one thing, don’t overload

“Which nodes?” is a selector

Use familiar tools and patterns
Jobs
**Run-to-completion**, as opposed to run-forever
- Express parallelism vs. required completions
- Workflow: restart on failure
- Build/test: don’t restart on failure

Aggregates success/failure counts

Built for batch and big-data work
StatefulSets
Goal: enable clustered software on Kubernetes

- mysql, redis, zookeeper, ...

Clustered apps need “identity” and sequencing guarantees

- stable hostname, available in DNS
- an ordinal index
- stable storage: linked to the ordinal & hostname
- discovery of peers for quorum
- startup/teardown ordering
ConfigMaps
**Goal: manage app configuration**

- ...without making overly-brittle container images

**12-factor** says config comes from the environment

  - Kubernetes is the environment

Manage config via the Kubernetes API

Inject config as a virtual volume into your Pods

  - late-binding, live-updated (atomic)
  - also available as env vars
Secrets
Goal: grant a pod access to a secured *something*  
• don’t put secrets in the container image!

12-factor says config comes from the environment  
• Kubernetes is the environment

Manage secrets via the Kubernetes API

Inject secrets as virtual volumes into your Pods  
• late-binding, tmpfs - never touches disk  
• also available as env vars
## Kubernetes Terminology

<table>
<thead>
<tr>
<th>Deployment</th>
<th>ReplicaSet</th>
<th>DaemonSet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pod</td>
<td>Liveness Probe</td>
<td>Job</td>
</tr>
<tr>
<td>Volume</td>
<td>Readiness Probe</td>
<td>StatefulSet</td>
</tr>
<tr>
<td>Label</td>
<td>Service</td>
<td>ConfigMap</td>
</tr>
<tr>
<td>Selector</td>
<td></td>
<td>Secret</td>
</tr>
</tbody>
</table>
There is more!
Thank You

Send talk feedback
bit.ly/atamel

Mete Atamel
@meteatamel
atamel@google.com
meteatamel.wordpress.com

kubernetes.io
cloud.google.com/container-engine