Fundamentals of Production Microservices

Owen Garrett
Deepfence, Inc.
owen@deepfence.io
Today’s Agenda

• Why Microservices
• Technology Introduction – Containers, Kubernetes, CI/CD
• Controlling Ingress Traffic with an Ingress Controller
• Controlling Internal Traffic with a Service Mesh
• Securing your Microservices Application
What is your organization’s expertise with Microservices?

1. What’s a Microservice? I’m here to learn
2. We’re not using a Microservices architecture yet.
3. We are taking first steps to production Microservices.
4. We run both Microservices and Traditional architectures in production.
5. We are (almost) entirely a Microservices-first organization.
Building Ships – Europe, c1500

Shipbuilding in Europe, c1500

Ships were built in-place, by hand, taking months to construct.

The guild system provided craftsmen and maintained a monopoly of skills and training. It protected workers and created artifacts of great quality but was slow and inefficient.
Building Ships – Venice, c1500

Shipbuilding in Venice, c1500

The Venetian Arsenal was the most powerful and efficient shipbuilding enterprise in the world.

It built a ship every day, with up to 100 galleys of various specification in the backlog.

In 1574, King Henry of France watched the outfitting of an entire ship during his lunch!
What has this got to do with Microservices?
Microservices architecture is an approach in which a single application is composed of many loosely coupled and independently deployable smaller services:

• Highly maintainable and testable
• Loosely coupled
• Independently deployable
• Organized around business capabilities
• Owned by a small team
What’s your biggest concern with microservices?

1. Getting Started: Training & Knowledge
2. In production: Logging, Visibility, Monitoring
3. In production: Security
4. In production: Scaling apps & teams
5. None: I know what I’m doing, all is working
Key Technologies for Production Microservices
Modern Apps and Modern Technologies

From Monolithic ...

- Three-tier, J2EE-style architectures
- Complex protocols (HTML, SOAP)
- Persistent deployments
- Fixed, static Infrastructure
- Big-bang releases
- Silo’ed teams (Dev, Test, Ops)

... to Microservices

- Microservices
- Lightweight (REST, JSON)
- Containers, VMs, Functions
- Infrastructure as Code; orchestration
- Continuous delivery
- DevOps Culture
What is a Container?

• A ‘container’ takes the bare minimum needed for an application, and packages it up as a single artifact:
  
  • Runnable application code – compiled, or needing a framework
  • Runtime artifacts – libraries, frameworks, ...
  • Declaration of runtime requirements – network, storage, ...

![Diagram of development lifecycle with Code, Container, Develop, Build, Package, Test, Deploy, Operate stages]
What is Kubernetes?

- Kubernetes creates a flat platform for running Containers (pods):
Fundamentals of Production Microservices

Continuous Integration

Dev

Commit

Build

Test

Push to Repo

Continuous Delivery

DevOps

Review

Deploy

Operations

Users

Infra LB

App LB

Prod 1

Staging

Prod 2

DR

Firewall

Auth

Monitor

Logs

Ops
Operating a distributed application is hard

<table>
<thead>
<tr>
<th>Static, Predictable Monolith:</th>
<th>Dynamic, Distributed App:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast, reliable function calls</td>
<td>Slow, unreliable API calls</td>
</tr>
<tr>
<td>Local debugging</td>
<td>Distributed fault finding</td>
</tr>
<tr>
<td>Local profiling</td>
<td>Distributed tracing</td>
</tr>
<tr>
<td>Calendared, big-bang upgrades</td>
<td>In-place dynamic updates</td>
</tr>
<tr>
<td>‘Integration hell’ contained in dev</td>
<td>‘Continuous Delivery’ live in prod</td>
</tr>
</tbody>
</table>

More things can go wrong, it’s harder to find the faults, everything happens live
Fundamentals of Production Microservices

Continuous Integration

Dev
Commit
Build
Test
Push to Repo

Continuous Delivery

Devops
Review
Deploy

Users
Infra LB
App LB

Staging
Prod 2
Prod 1
DR

Operations

Ops
Firewall
Auth
Monitor
Logs
Fundamentals of Production Microservices

- Ingress Controller
- Service Mesh

Application Dataplane
Begin with the Ingress Controller

- How do I get traffic to my services in Kubernetes?
- How can I do this in a secure, reliable and self-serve manner?
Two Challenges at Scale

Complex Applications
How can you provide the advanced capabilities that complex applications require?

Multitenancy (Teams)
How can multiple teams and applications share a Container environment safely and securely?
What Ingress solution do you use?

1. Default Kubernetes Ingress Controller
2. Default OpenShift Router
3. Third-party Ingress Controller (which one?)
4. Don’t know
The Ingress Controller

A specialized load balancer for Kubernetes environments:

- Accepts traffic from outside Kubernetes, and load-balances it to pods (containers) running inside Kubernetes
- Configured using the Kubernetes API, with objects called ‘Ingress Resources’
- Monitors the pods running in Kubernetes, and automatically updates the load balancing rules if, for example, pods are added or removed from a service
## Comparison of Kubernetes Ingress controllers

<table>
<thead>
<tr>
<th></th>
<th>nginx</th>
<th>nginx</th>
<th>nginx</th>
<th>nginx + Azure App Gateway</th>
<th>HAPProxy</th>
<th>HAPProxy (Sovereign)</th>
<th>Voyager</th>
<th>Istio Ingress</th>
<th>Contour</th>
<th>Ambassador</th>
<th>Gloo</th>
<th>Traefik</th>
<th>Skipper</th>
<th>Citrix Ingress</th>
<th>GKE Ingress</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Path</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Headeds</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Querystring</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hosts</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Needs help</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Notes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LearnK8s Research**

- [https://docs.google.com/spreadsheets/d/191WWNpjJ2za6-nbG4ZoUMXMrUK8KICosvQB0f-qq3k/edit#gid=907731238](https://docs.google.com/spreadsheets/d/191WWNpjJ2za6-nbG4ZoUMXMrUK8KICosvQB0f-qq3k/edit#gid=907731238)

---

**Notes**

- Leave a comment or drop us a line at research@learnk8s.io
- License:
  - See [LearnK8s Research](https://docs.google.com/spreadsheets/d/191WWNpjJ2za6-nbG4ZoUMXMrUK8KICosvQB0f-qq3k/edit#gid=907731238)
- Last updated: February 17, 2021

Find more research at [LearnK8s Research](https://docs.google.com/spreadsheets/d/191WWNpjJ2za6-nbG4ZoUMXMrUK8KICosvQB0f-qq3k/edit#gid=907731238)
Kubernetes Ingress Resources

- Simple yaml specification of an ingress requirement:
  - Hostname
  - Path
  - Service
  - TLS requirements

- Ingress Controller uses these resources to configure the load balancer
  - Customized using annotations, configmaps ...

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: simple-fanout-example
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        pathType: Prefix
        backend:
          service:
            name: service1
            port:
              number: 4200
      - path: /bar
        pathType: Prefix
        backend:
          service:
            name: service2
            port:
              number: 8080
```
NGINX Ingress Resources – example of a Domain-Specific Language

Host
- TLS
- Policies
- Upstreams
- Routes
  - Path
  - Policies
  - Action
  - Split
  - Match
  - Route
  - ErrorPage

VirtualServer

VirtualServerRoute

NGINX server configuration
NGINX http configuration
Server and HTTP snippets

Access Control
Rate Limiting
Auth (JWT, OIDC)
MTLS (Ingress/Egress)
App Protect WAF

Policies

Location snippets

NGINX location configuration
Other Implementations are emerging

Evolving Kubernetes networking with the Gateway API

Thursday, April 22, 2021

Authors: Mark Church (Google), Harry Bagdi (Kong), Dannyon Hanson (Red Hat), Nick Young (VMware), Manuel Zapf (Traefik Labs)

The Ingress resource is one of the many Kubernetes success stories, it created a diverse ecosystem of ingress controllers which were used across hundreds of thousands of clusters in a standardized and consistent way. This standardization helped users adopt Kubernetes. However, five years after the creation of Ingress, there are signs of fragmentation into different but strikingly similar CRDs and overloaded annotations. The same portability that made Ingress pervasive also limited its future.

It was at KubeCon 2019 San Diego when a passionate group of contributors gathered to discuss the evolution of Ingress. The discussion overflowed to the hotel lobby across the street and what came out of it would later be known as the Gateway API. This discussion was based on a few key assumptions:

1. The API standards underlying route matching, traffic management, and service exposure are commoditized and provide little value to their implementers and users as custom APIs
2. It’s possible to represent L4/L7 routing and traffic management
Spotlight on Use Cases

- **Traffic Steering**: Multiple versions, multiple clients
- **Split Routing**: A/B Testing for safe production deployment
- **Error Pages**: Implement ‘Circuit Breakers’ to contain failures
- **Rate Limiting**: Protect vulnerable apps, limit greedy clients
- **Authentication**: Offload and centralize identity checking
- **WAF**: Block incoming traffic
- **NGINX Snippets**: “to-the-metal” with NGINX configuration
Service Mesh

• Many operational challenges happen *within* the application
• How do I secure, observe and manage traffic inside my Kubernetes environment?
North-South and East-West Traffic

"North-South" traffic
Ingress Controller

"East-West" traffic - Service Mesh
Do you use a Service Mesh in production?

1. No, and I’m not planning to use one yet
2. No, but I’m actively evaluating
3. Yes – Istio in production
4. Yes – a different mesh in production (which one?)
5. Don’t know
What Is A Service Mesh?

Service Mesh aims to improve application traffic control, observability and security for distributed systems.

- The New Stack

Secure Traffic
End-to-end encryption (Mutual TLS / mTLS), ACLs

Manage All Service Traffic
Load Balance, Circuit breaker, B|G, Rate Limiting…

Measure Traffic
Generate transaction traces and real-time monitoring

Copyright 2021 Deepfence, Inc.
## Comparison of service meshes

<table>
<thead>
<tr>
<th></th>
<th>Istio</th>
<th>Linkerd2</th>
<th>Kuma</th>
<th>Consul connect</th>
<th>AWS App Mesh</th>
<th>NGINX Service Mesh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Google, IBM, Lyft</td>
<td>Go / Rust</td>
<td>Kong</td>
<td>Go</td>
<td>Amazon</td>
<td>Go / C</td>
</tr>
<tr>
<td>License</td>
<td>Apache License 2.0</td>
<td>Apache License 2.0</td>
<td>Mozilla License</td>
<td>Apache License 2.0</td>
<td>Closed source</td>
<td>Closed source / Apache License</td>
</tr>
<tr>
<td>Built-in, Envoy</td>
<td>Envoy</td>
<td>Envoy</td>
<td>Built-in, Envoy</td>
<td>Envoy</td>
<td>Envoy</td>
<td>Envoy</td>
</tr>
<tr>
<td>Notes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Notes
- How to contribute: Leave a comment or drop us a line at research@learnk8s.io
- Last updated: February 2, 2021

Find more research at [https://learnk8s.io/research](https://learnk8s.io/research)
Spotlight on Use Cases

- **Mutual TLS**: Secure traffic in a zero-trust network
- **Instrumentation**: Monitor performance, latency
- **Tracing**: Debug transactions and locate faults
- **Rate Limiting**: Protect apps, limit greedy clients
- **Traffic Splitting**: Implement Canary, Blue-Green upgrades
- **Access Control**: Implement allow-lists to govern traffic
- **Egress Control**: Manage and broker outbound traffic
When Am I Ready For A Service Mesh?

- You are fully invested in microservices and using Kubernetes
- You are GitOps ready, with a fully-automated CI/CD pipeline
- You are deploying frequently to production (at least once per day)
- Your applications are complex, in number of services and in depth
- You need mTLS in a zero-trust production environment
- You have a learning culture
Securing your Microservice Application
Why are Microservices Apps so challenging to secure?

Highly-distributed
Rely on fast-moving open source

| 528 | Average # of OSS components in a modern application
| 4 years | Average detection time for vulnerability in existing code
| 18,000+ | New CVEs published per year, growing year-on-year |

Very broad, hard-to-secure attack surface
Securing Microservices Apps

Pre-Deployment
- Code Analysis
- Vulnerability Detection
- CSPM (compliance)

In Production
- Web App Firewall
- Runtime Detection

Incident Response
- Runtime Protection
- Post-Event Forensics
Securing Microservices Apps

- **Pre-Deployment**
  - Code Analysis
  - Vulnerability Detection
  - CSPM (compliance)

- **In Production**
  - Web App Firewall
  - Runtime Detection

- **Incident Response**
  - Runtime Protection
  - Post-Event Forensics
Review
What have we learnt?

- Microservices is a long journey, not a quick destination
- Begin when the need to iterate and improve is high
- Expect to encounter Containers and Kubernetes
- Be able to control the data plane
- Be ready for a new Security Landscape!