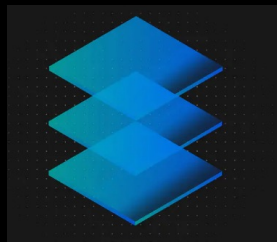
Build your Microservices with IBM Kubernetes & Istio

IBM | Red Hat



Agenda

- Hands-on preparation 10 min
- Cloud-native development and Istio – presentation 20 min
- Hands-on / Demo – 1:20
- Quiz 10 min

- Total time: 2 hours

https://github.com/agavrin/cloud-native-workshop-2021

Check your email for credentials

IBM Workshops <ibmcloudcoupon.noreply@gmail.com>

кому: я

Hello A!

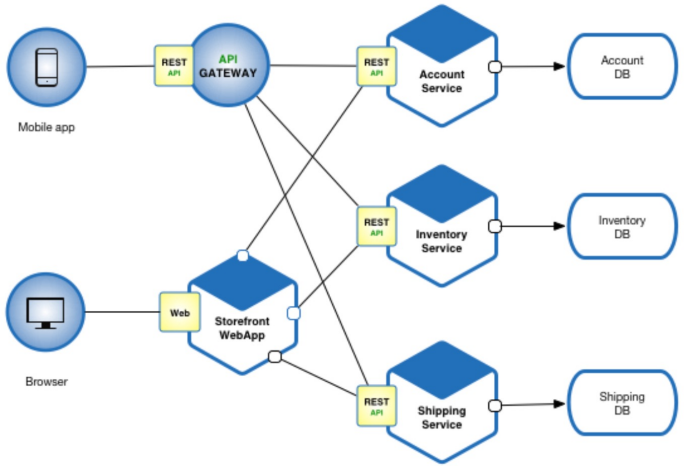
Please use the following assigned resource:

server=158.177.16.131
userid=user1
password=8MPVQyhR63xao

Any questions - please contact: agavrin@ru.ibm.com

Characteristics of **cloud-native** applications

- Applications that adapt to the Cloud (scalability)
- Supporting a large range of devices and user interfaces
- Automated (provision-deploy-scale)
- CI/CD (agility)
- Support multiple datastore
- APIs at the heart of the applications
- Microservices



Rules for developing/moving applications to the cloud

1. Don’t code your application directly to a specific topology
2. Don’t assume the local file system is permanent
3. Don’t keep session state in your application
4. Don’t log to the file system
5. Don’t assume any specific infrastructure dependency
6. Don’t use infrastructure APIs from within your application
7. Don’t use obscure protocols
8. Don’t rely on OS-specific features
9. Don’t manually install your application

Cloud-Native Application Goals – Day1

Horizontal scaling

- Application runs in multiple runtimes spread across multiple hosts (VIII)

Immutable deployment

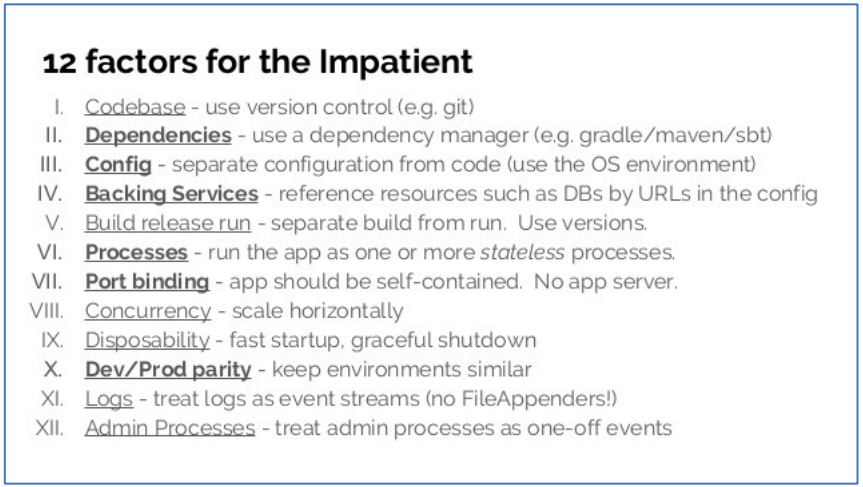
- A runtime is not patched, it's replaced (IX)
- A runtime is stateless (VI)
- Shared functionality in backing services (IV)

Elasticity

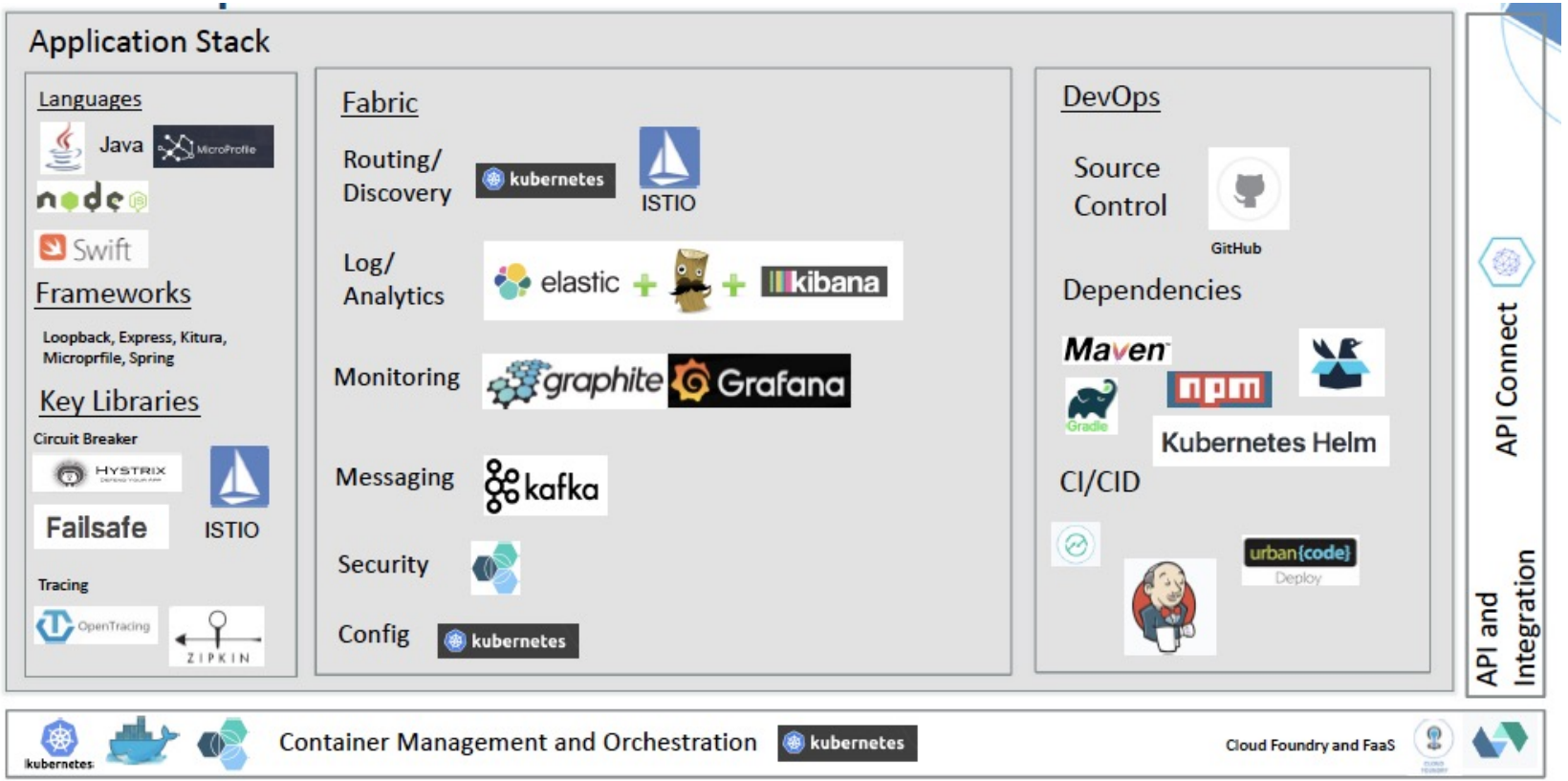
- Automatic scale-out and scale-in to maintain performance
- Achieved via containerization

Pay-as-you-go charging model

- Pay for what you use



Development Stack - Choices



Microservices: Making developers more efficient

- An **engineering approach** that reduces an application into **single-function** modules
- They have well-defined **interfaces** that are independently deployed

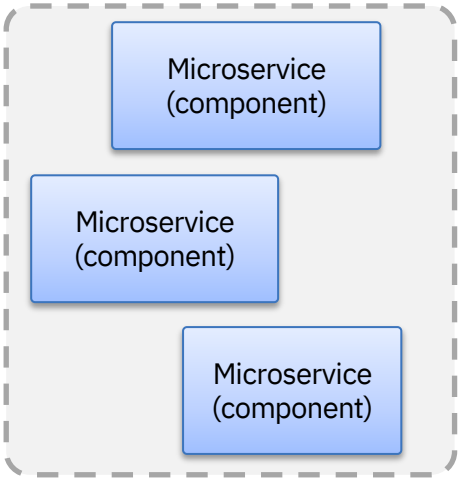
- They are operated by a **small team** which owns the entire **lifecycle** of the service
- Microservices accelerate delivery by

- minimizing communication and coordination between people
- reducing the scope and risk of change

Microservices Architecture ?

Simplistically, microservices architecture is about breaking down large silo applications into more manageable fully decoupled pieces

Monolithic application



Microservices application

Agility
Scalability
Resilience

A microservice is a granular decoupled component within a broader application

Why Microservices?

Small scoped, independent, scalable components

Scaling

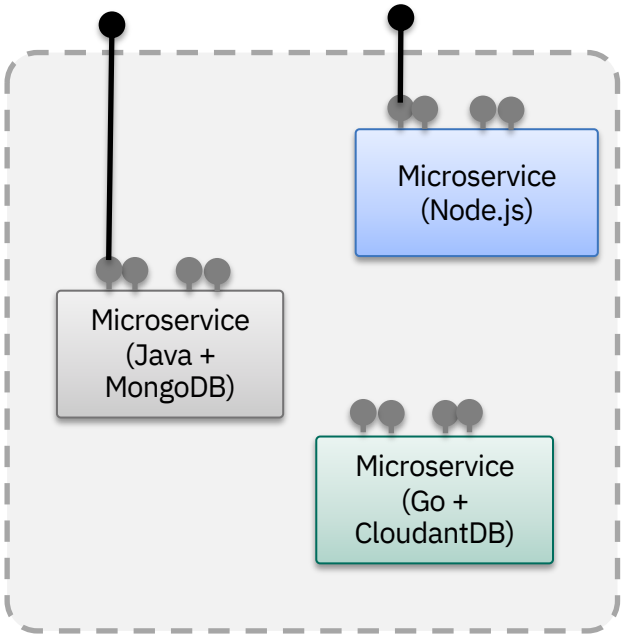
- Elastic scalability
- Workload orchestration

Agility

- Faster iteration cycles
- Bounded context (code and data)

Resilience

- Reduced dependencies
- Fail fast



Microservices application

Microservices inter-communication

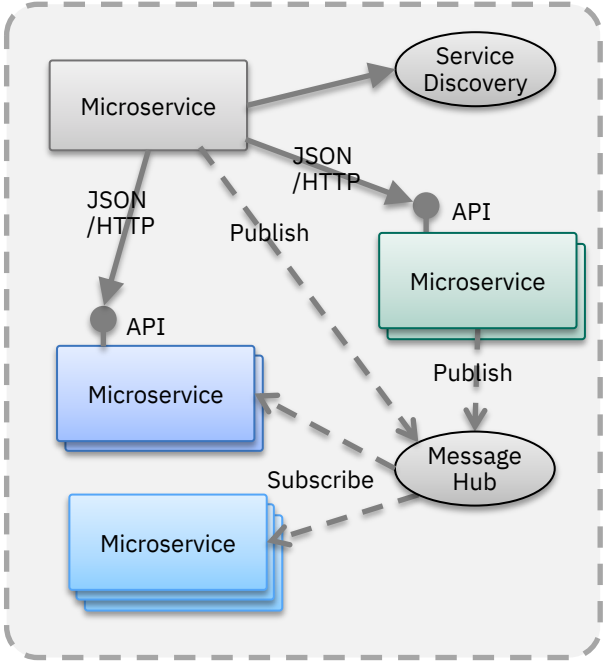
Aim is decoupling for robustness

Compose a complex application using

- “small”
- independent (autonomous)
- replaceable
- processes

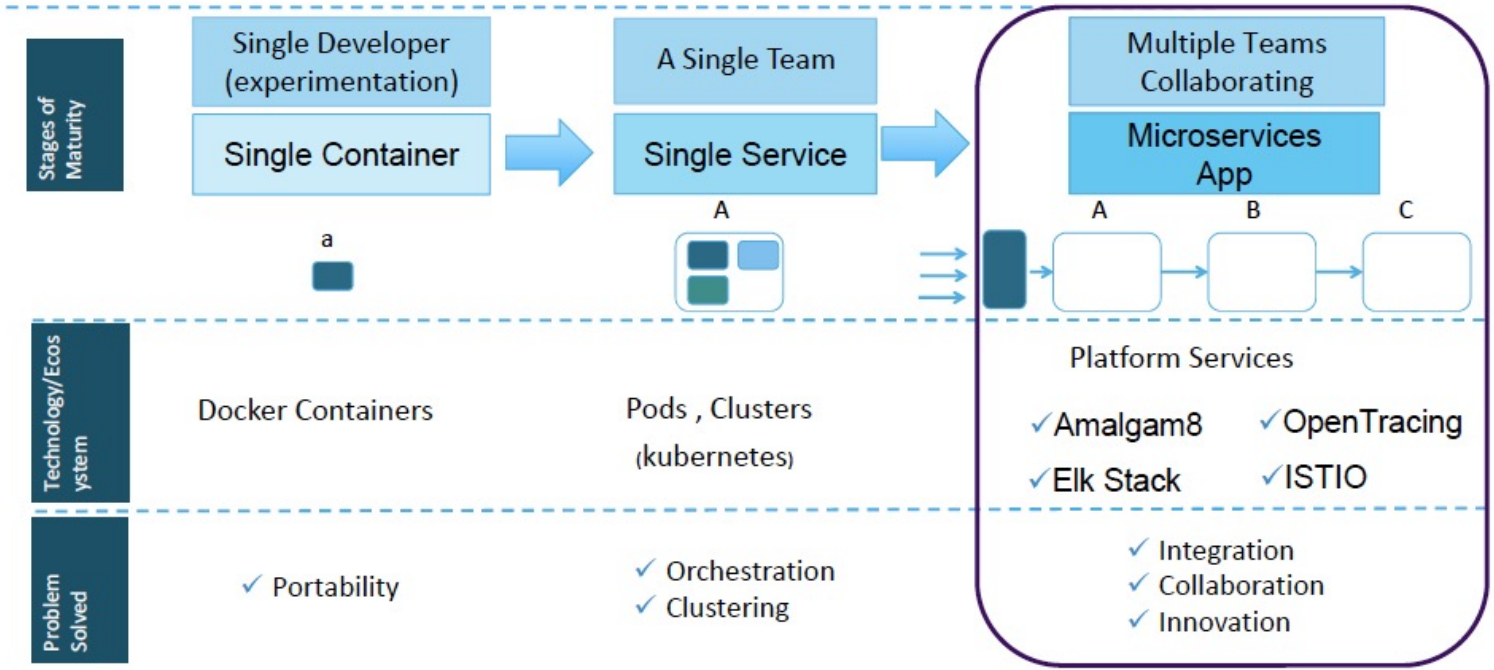
that communicate

- via language-agnostic APIs
- synchronously and asynchronously



Microservices application

Microservices: Built for the enterprise journey



Advantages / Challenges of Microservices

Advantages

- Developed independently
- Developed by a single team
- Developed on its own timetable
- Each can be developed in a different language
- Manages its own data
- Scales and fails independently

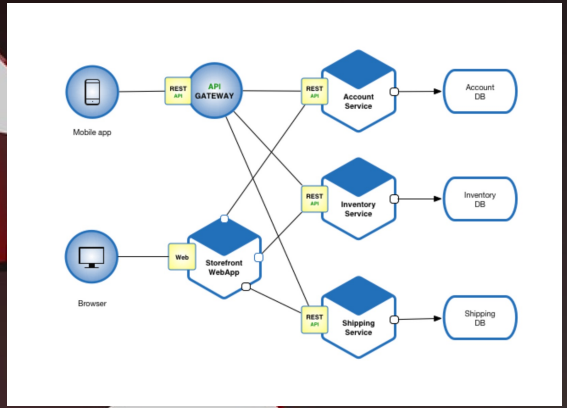
Challenges

- Developers must have significant operational skills (DevOps)
- Service interfaces and versions
- Duplication of effort across service implementations
- Extra complexity of creating a distributed system:

Designing decoupled, non-transactional systems is difficult
Locating service instances
Maintaining availability and consistency with partitioned data
End-to-end testing

Things to take care of

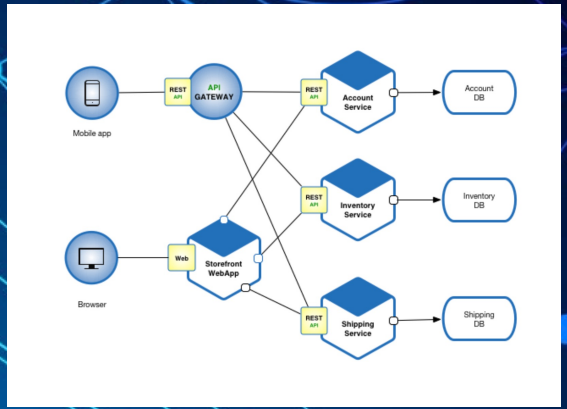
- Service discovery
- Load balancing
- Fault Tolerance
- Monitoring
- Dynamic routing
- New versions
- Security



What is a ‘Service Mesh’ ?

A network for services

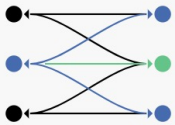
- Observability
- Resiliency
- Traffic Control
- Security
- Policy Enforcement
- Zero code change



Istio, the service mesh – Day 2

Istio

Connect, secure, control, and observe services.

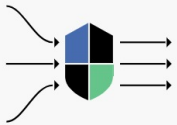


Connect

Intelligently control the flow of traffic and API calls between services, conduct a range of tests, and upgrade gradually with red/black deployments.

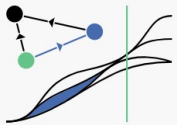
Secure

Automatically secure your services through managed authentication, authorization, and encryption of communication between services.



Control

Apply policies and ensure that they're enforced, and that resources are fairly distributed among consumers.

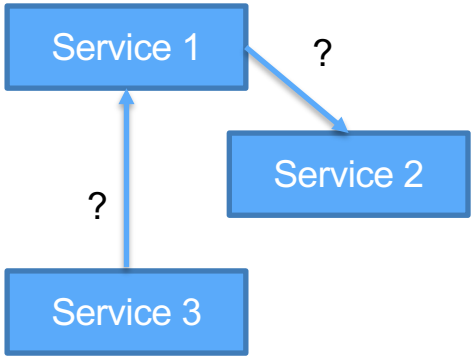


Observe

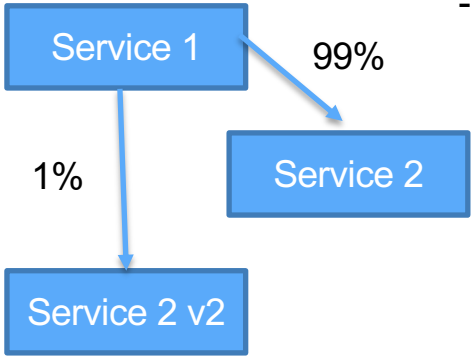
See what's happening with rich automatic tracing, monitoring, and logging of all your services.

- IBM
- Lyft
- Google
- Some others

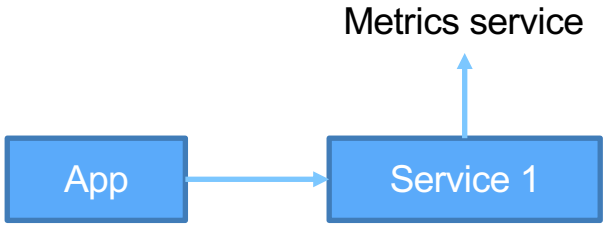
What is Service Mesh?



- Authorization



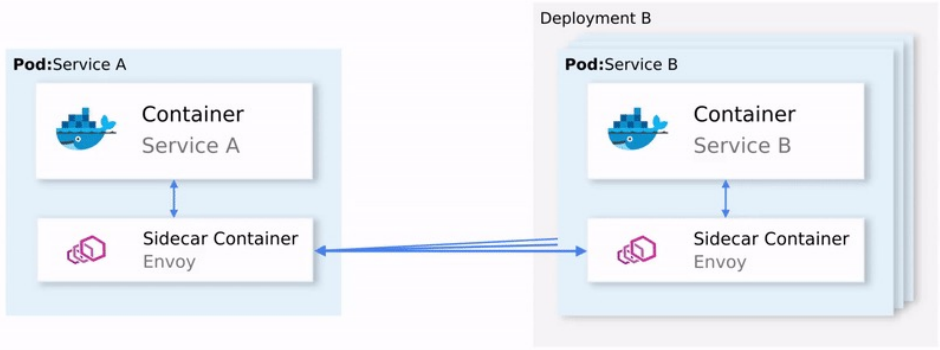
- Canary deployments



- Metrics

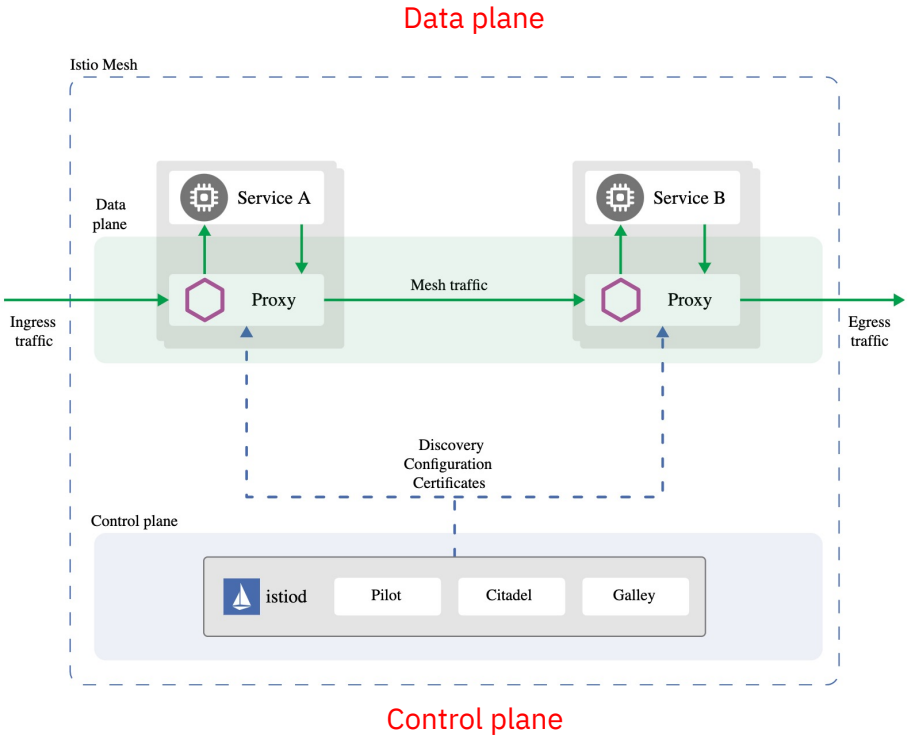
Envoy : the sidecar proxy in each POD

- Dynamic service discovery
- Load balancing
- TLS termination
- HTTP/2 and gRPC proxies
- Circuit breakers
- Health checks
- Staged rollouts with %-based traffic split
- Fault injection
- Rich metrics



https://medium.com/google-cloud/back-to-microservices-with-istio-p1-827c872daa53

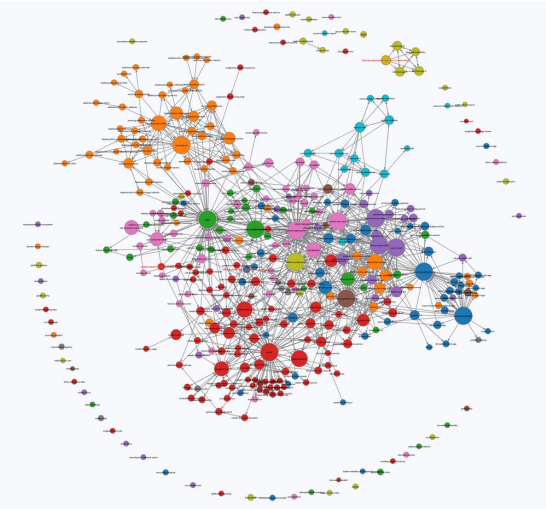
Istio Architecture



- The **data plane** is composed of a set of intelligent proxies (Envoy) deployed as sidecars. These proxies mediate and control all network communication between microservices.
- The **control plane** manages and configures the proxies to route traffic.

Performance and Scalability (Istio 1.9.5)

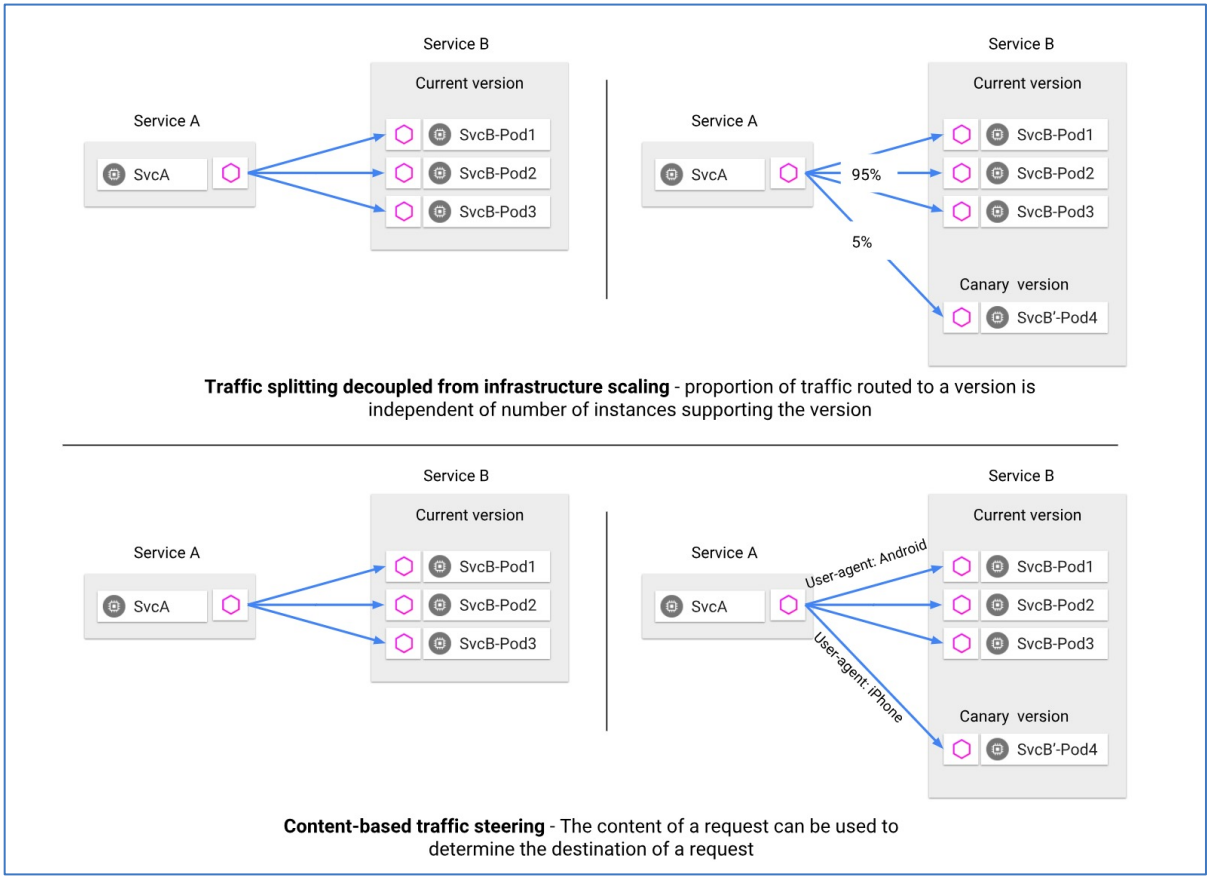
- 1000 services, 2000 sidecars
- 70000 requests per second



- **Envoy** proxy uses **0.35 vCPU** and **40 MB** memory per 1000 requests per second
- **Envoy** proxy adds **2.65 ms** to the 90th percentile latency.
- **Istiod** uses **1 vCPU** and 1.5 GB of memory.
- **Istiod** can be scaled hotizontally

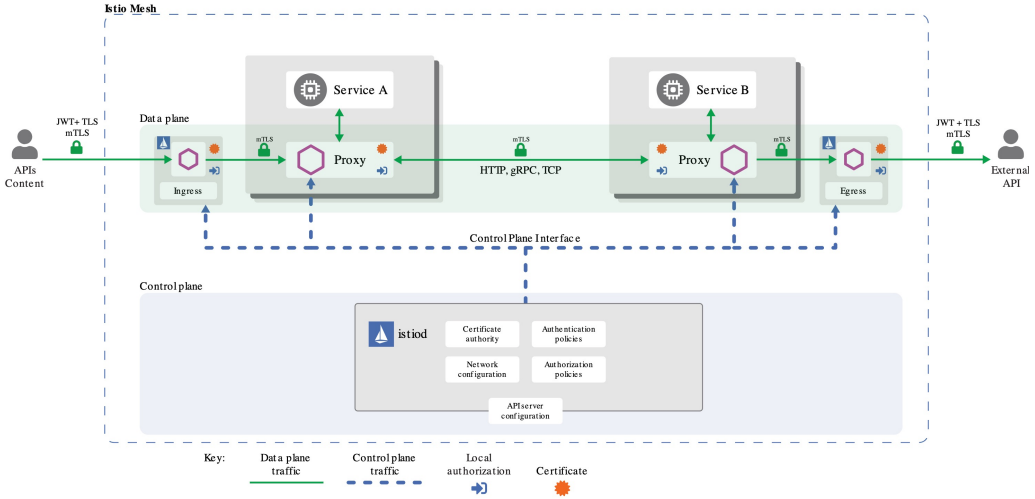
https://istio.io/latest/docs/ops/deployment/performance-and-scalability/

Traffic Management



- Request Routing
- Fault Injection
- Traffic Shifting
- TCP Traffic Shifting
- Request Timeouts
- Circuit Breaking
- Mirroring
- Rate limits

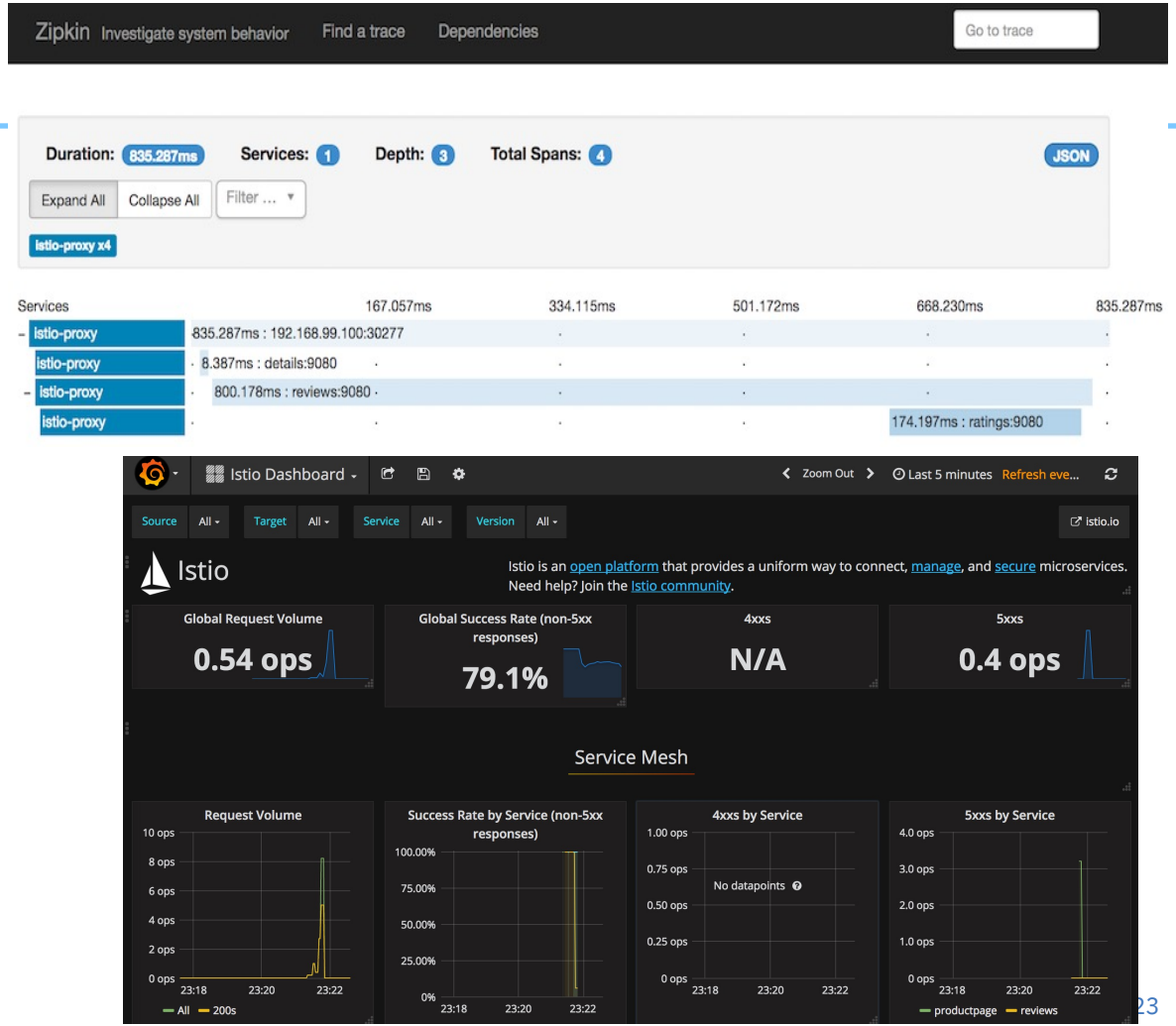
Security



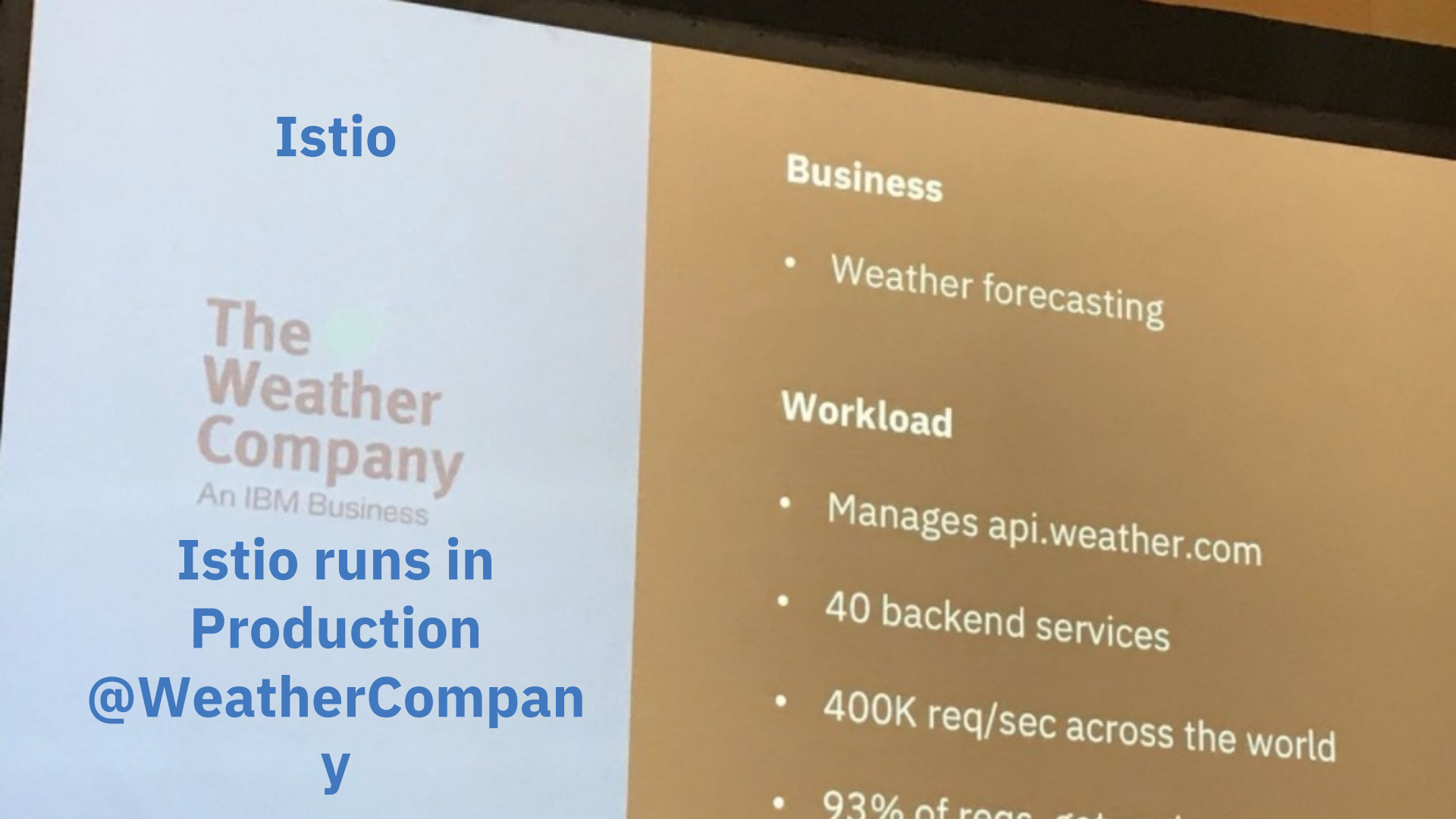
- Certificate Management
- Authentication
- Authorization

Observability

- Metrics
- Logs
- Distributed Tracing



Istio



Istio runs in Production @WeatherCompany