DevOps 2.0

Using Modern Tools and Practices to Develop, Maintain, and Manage Scalable Microservices

Joe McCormick, Architect
Boeing
Joe McCormick has more than 25 years of software experience in companies ranging from Dot Com startups to large Fortune 100 companies, usually filling roles in Development, Architecture, and Development Management. He has extensive experience in designing, using, creating, and implementing Software Configuration Management systems, build and deployment systems, Application Lifecycle Management tools, and other software delivery pipeline enablers, making him an expert in the concepts of Continuous Integration and Delivery, Agile development processes, DevOps, and Service-Oriented Architecture concepts like Microservices.

The former Long Island, NY Firefighter, Emergency Medical Technician, and Army Crew Chief (OH-6, OH-58, UH-1, and UH-60 helicopter airframes) now lives in Charleston, SC and races sailboats in his free time (this will become evident during the following presentation).

Joe is currently working as an Architect in Boeing's Future State Technology Architecture group under the Enterprise Architecture organization of Information Technology.
Overview

Core Concepts

- Microservices
- DevOps
- Containers
- Blue Green Deployments
- Agile
- Lightweight APIs
- Proxy Services
- Logging & Analytics
- Discovery
- Self Healing
- Continuous Monitoring & Response
- Automated Scaling
- Orchestration
- Test Driven Development
- Continuous Delivery
- RESTful JSON
Request: This presentation is an overview and integration of more than a few concepts, many of which could be presented on their own (or could even be their own workshops or courses). The first third of this presentation contains definitions and overviews of high-level ideas that will be covered quickly and the details will be covered in later slides. Please hold your questions until solicited. Thank You.

Note: If you are viewing this material on your own (i.e. Joe is not presenting the deck), please view this in “presentation mode”. As a time and slide saving measure, animations are used throughout the presentation and the slides will not render correctly (or will not be readable at all) if you are not in “presentation mode”.

DISCLAIMER: The software and tools used in this presentation are for conceptual demonstration and do not represent the standard tools and development patterns of any organization or company. Please consult YOUR organization for standard tooling, patterns, processes, and best practices.
Our Example Application: Sailboat Management
Our Test Application: Sailboat Management

User Story:
As a Boat Manager, I need to see the age of each crewmate so that the company can better plan for the Youth Sailing Program events and races.

• Task: Add an age text field on the crew information page.
  • Status: Completed
  • Comment: Implemented, but is Feature Flagged to not show until the crew-assignments service is updated.

• Task: Update the data model of the crew-assignments service to accept age as a new ‘float’ property.
  • Status: Not Yet Started
crew-assignments Service Description

Global Product Data Interoperability Summit | 2016

Positions
- positionId {id}
- name: string
- location: string
- assigned: collection<model>

Crew
- crewId {id}
- yearsExperience: number
- position: model
- age: number

PositionsController
- position: model
- createPosition(position): number
- updatePosition(position): number
- getPositions(): number
- getPosition(position.id): number
- deletePosition(position.id): number

CrewController
- crew: model
- createCrewmate(crew, positionId): number
- updateCrewmate(crew, positionId): number
- getCrew(): number
- getCrewmate(crew.id, positionId): number
- deleteCrewmate(crew.id): number

36 lines of code
244 Lines of test code
0 lines of code
crew-assignments Development Environment
(“DevOps in a Box”)
Provisioned Environment

```bash
$ cd DevopsInABox/CommonDevInf/
~/DevopsInABox/CommonDevInf
$ vagrant up dev --provision
```

<table>
<thead>
<tr>
<th>real</th>
<th>11m47.712s</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>0m0.031s</td>
</tr>
<tr>
<td>sys</td>
<td>0m0.110s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>real</th>
<th>11m49.978s</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>0m0.015s</td>
</tr>
<tr>
<td>sys</td>
<td>0m0.093s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>real</th>
<th>4m12.002s</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>0m0.000s</td>
</tr>
<tr>
<td>sys</td>
<td>0m0.031s</td>
</tr>
</tbody>
</table>
Gartner DevOps Model (Gartner 2015)
The Build Pipeline

Global Product Data Interoperability Summit | 2016
crew-assignments Docker Layers

**Crew-assignments-tests**
Adds front-end, wget, xvfb, mongodb, firefox, chrome, selenium, npm installs dependencies like mocha, should.js, supertest

**Crew-assignments**
Adds the crew-assignments service

**Sailsbase**
Adds Sails via npm (and runtime dependencies)

**Google/nodejs**
Adds curl, python, git, nodejs/npm

**Debian/Wheezy**
Base image

**Testing image**

**Service runtime**

**Service base image**

**Public base image**

Docker Hub

Docker private registry

SuperTest

sails

node.js

npm

debian

MOCHA

SHOULD JS

GLOBAL PRODUCT DATA INTEROPERABILITY SUMMIT 2016
Managing a Single Service on Multiple Nodes

http://crew-assignments

Jenkins Pipeline Deploys to the Swarm Master

Swarm determines where the service should go

Registrar sees the deployment and notes relevant information

Consul updates our reverse proxy, consul instances on nodes and updates / creates service checks

Nginx is now our gateway with our predetermined ip, name, and / or port (and can do more than we show – like ssl)
crew-assignments Runtime Environment
What our Setup Can Do

Zero-Downtime Deployments

Know the health of every node and every container in those nodes

Plan for Load
React to Load

Automatically recover when services stop responding (or even when nodes or entire datacenters stop responding)
Scaling (X-Axis)

Global Product Data Interoperability Summit | 2016

crew-assignments-blue-8080

TAGS
No tags

NODES

- **prod-2** 10.22.22.152  3 passing
  - Disk utilization: **disk**  passing
  - Memory utilization: **mem**  passing
  - Serf Health Status: **serfHealth**  passing

- **consul**  3 passing
- **crew-assignments**  4 passing
- **crew-assignments-blue-8080**  3 passing
- **nginx-80**  3 passing
- **swarm-master**  3 passing
Scaling and Descaling
Rollback to Previous Runtime

Global Product Data Interoperability Summit | 2016

Dashboard [jenkins]

Filter by name: any status

crew-assignments-blue-8080

TAGS
No tags

NODES

prod-1 10.22.22.151
Disk utilization disk
Memory utilization mem
Sorf Health Status serfHealth
3 passing

prod-2 10.22.22.152
Disk utilization disk
Memory utilization mem
Sorf Health Status serfHealth
3 passing
Logging and Analytics

Logging & Analytics

Predict and Avoid

Data Visualization

Data Analytics

Data Collection (Centralized Logging)

Our setup has all three preconfigured (via Ansible) and each is running in a container
• Of course we are! DevOps *must include* Continuous Improvement
  • Our example database is running in one container. We need to apply X-Axis scaling to our Z-Axis solution
  • Our Build Pipeline does not include any static analysis or security testing
  • We have a HUGE architectural problem: Our aggregation layer itself is not redundant or scaled
    – Thankfully, Docker Swarm, Consul, and Nginx all support clustering themselves
  • We need ALM Integration
  • Maybe implement an enterprise service registry so we can find service available to develop against? Maybe not so that we keep coupling loose.
• More?
Microservices: Not for Everything

Note: Adapted from Gartner (2014)
Summary – Why Microservices?

- Leverage DevOps concepts and tooling to drastically decrease release cycle time so much so that Continuous Deployment can be implemented.
- Infrastructure and Tooling can be easily replicated.
- Development teams are small – maybe even one developer (but, there are others involved – Architect, QA, System Administrators, etc.).
- Containers ensure all parts of an application are developed, tested, and deployed via the same process and that the service can run anywhere (bare metal, VM, cloud).
- They can be scaled (up or down) very easily – even automated.
- Monitoring can not only detect issues, but actually heal the system or prevent an issue from ever happening.

Note: Adapted from Gartner (2014)
Additional Resources and Informational Slides

• Some of the tooling and setup of the demonstrations used in the presentation were modified after reading the book *The DevOps 2.0 Toolkit* by Viktor Farcic (Farcic, 2016). I highly recommend this book for anyone who wants a hands-on look at these concepts.

• For a better understanding of scaling, scalability, and related concepts, a great resource is *The Art of Scalability* (Abbott and Fisher, 2015).

• Boeing personnel please look for us on inSite.

Note: Adapted from Gartner (2014)
What are Microservices?

• Definitions
  • Microservices are a more concrete and modern interpretation of service-oriented architectures (SOA) used to build distributed software systems. It is an architectural style that is a first realization of SOA after the introduction of DevOps and this is becoming the standard for building continuously deployed systems. (Microservices, August 9, 2016)

  • Microservices are an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery. (Fowler, 2014)

Company list source: Richardson (2014)
Microservices Characteristics

- **Quickly** developed
  - Speed over elegance
- **Stateless**
- Designed for **Failure**
  - Netflix Simian Army
- **Elastic**
- Easily **replaceable**
- Use **Continuous Delivery**
- **Modular** in structure
- **Independently** deployable
- Technology **agnostic**
- **Finely-grained**
  - "*Do one thing and do it well*" (Unix philosophy, July 31, 2016)
- Typically implemented via **APIs** (Application Programming Interface) over **HTTP/REST** (Representational State Transfer) using **JSON** (JavaScript Object Notation)
  - Not a requirement
What is DevOps?

Gartner’s Definition of DevOps (Gartner, 2014):

- “a change in IT culture, focusing on rapid IT service delivery through the adoption of agile, lean practices in the context of a system-oriented approach.”
- “emphasizes people (and culture), and seeks to improve collaboration between operations and development teams. Implementations utilize technology - especially automation tools that can leverage an increasingly programmable and dynamic infrastructure from a lifecycle perspective.”

Gartner further identifies 5 primary principles that underpin DevOps (Gartner, 2015):

- **Iterative**: well aligned with uncertainty; exhaustive planning is not optimal.
- **Continuous**: delivery & deployment; optimizing/experimenting with new processes, tools and org structures.
- **Collaborative**: agreement on the mission and metrics; transparent and frequent communications.
- **Systemic**: Agile initiatives not just focused on development, but downstream operations.
- **Automated**: technologic facilitator to deliver speed and scale with human involvement only by exception.
What are Containers?

Characteristics of Containers

- **Build once**, run anything anywhere
  - Completely portable -- no inconsistencies between development, test, production, or customer environments
- **Complete**
  - Dependent libraries and binaries
  - Configuration files
  - Middleware
  - Environment changes are built with the code and not as a separate process
- **Immutable**
  - No more “it ran fine on my box”, debug production issues using the production image in another environment.
  - Simpler scaling (X axis)
  - Enabler for self healing
- **Lightweight**
  - Easy to store, retrieve, change, deploy, and redeploy
  - Lower cost and higher performance than VMs alone.

A container is an isolated user-space virtualization instance. Think of them as managed chroot jails.

Containers are **isolated** but **share** OS and binaries and libraries where appropriate.

The result is significantly **faster** deployment, **less** overhead, **easier** migration, and **quicker** restarts.
Three Dimensional Scaling

X-Axis
- What most think of as scaling
- Clones running behind load balancers
- Can be resource intensive
  - Especially with large applications

Y-Axis
- Scale by splitting
- Typically implemented via SOA and microservices

Z-Axis
- Scale by partitioning data
- No centralized monolithic database system
- Each piece of an application is “responsible” for its own data

Adapted from Abbott and Fisher (2015)
<table>
<thead>
<tr>
<th>Category</th>
<th>Microservice</th>
<th>Traditional SOA</th>
<th>Monolith</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical Lines of Code</td>
<td>Typically less than 100</td>
<td>Hundreds to Thousands</td>
<td>Thousands to Millions</td>
</tr>
<tr>
<td>Data Model</td>
<td>NoSQL or Small SQL databases with existing RDBMS</td>
<td>Large RDBMS</td>
<td>Large RDBMS</td>
</tr>
<tr>
<td>Communication</td>
<td>Fast, lightweight, asynchronous messaging</td>
<td>Enterprise Service Bus, synchronous connections</td>
<td>N/A</td>
</tr>
<tr>
<td>Development Team</td>
<td>Very Small – possibly a single Developer</td>
<td>Normal Development teams, each focusing on one area.</td>
<td>Large teams of teams, with institutional knowledge</td>
</tr>
<tr>
<td>System Changes</td>
<td>Create a new service, abandon the old one</td>
<td>Modify existing services and architecture</td>
<td>Requires more architectural analysis, knowledge of large code bases, and seasoned Developers</td>
</tr>
<tr>
<td>Release Schedule</td>
<td>Continuous Delivery</td>
<td>Weeks to Months, coordination needed</td>
<td>Long cycles, Blockpoints</td>
</tr>
<tr>
<td>Scaling</td>
<td>Scales well X, Y, and Z axis</td>
<td>Scales in X, limited Y and Z axis</td>
<td>Difficult to scale in X axis, No Y axis scaling, Limited Z</td>
</tr>
</tbody>
</table>
Orchestration and Managing More than One Node

Global Product Data Interoperability Summit | 2016

Here’s what we don’t want: The maintenance nightmare of keeping up with deployment scripts and configurations in each of our services, in multiple nodes, especially since they may need to know about each other.

We need something to manage a “cluster” of nodes for us

“Docker Swarm is native clustering for Docker. It turns a pool of Docker hosts into a single, virtual Docker host.” (Docker, n.d.)

This is the first part of our scaling (more to come), so how do we implement basic X-Axis scaling?

• Runs on the aggregation layer
• Is a Docker Container
• The CD Pipeline (Jenkins) has a step to make sure it is running and available (using Ansible)
• Installed as a Swarm Master and Swarm nodes
Consul has multiple components, but as a whole, it is a tool for discovering and configuring services in your infrastructure. It provides several key features:

- Service Discovery
- Health Checking
- Key/Value Store
- Multi Datacenter

NGINX is a free, open-source, high-performance HTTP server and reverse proxy, as well as an IMAP/POP3 proxy server. NGINX is known for its high performance, stability, rich feature set, simple configuration, and low resource consumption. (Nginx, n.d.)

Registrar automatically registers and deregisters services for any Docker container by inspecting containers as they come online. (Gliderlabs n.d.)

Our Nginx runs in a Docker container.
Production Overview

Global Product Data Interoperability Summit | 2016

Diagram of production overview with various components:
- Aggregator
- Orchestration
- Proxy Services
- Automated Scaling
- Logging & Analytics
- Discovery
- Self Healing
- Blue Green Deployments
- Continuous Monitoring & Response
References


References

Global Product Data Interoperability Summit | 2016


Farcic, Viktor. (July 20, 2016). Devops 2.0. Automating the continuous deployment pipeline with containerized microservices. CreateSpace Independent Publishing Platform

All statements in this report attributable to Gartner represent Boeing’s interpretation of data, research opinion or viewpoints published as part of a syndicated subscription service by Gartner, Inc., and have not been reviewed by Gartner. Each Gartner publication speaks as of its original publication date (and not as of the date of this presentation/). The opinions expressed in Gartner publications are not representations of fact, and are subject to change without notice.