DevOps: A Software Architect's Perspective

Dr. Liming Zhu | Research Director
Oct 2015

www.csiro.au
NICTA + DP@CSIRO = Data61

• NICTA (National ICT Australia): Australia’s National Centre of Excellence in ICT
• Digital Productivity Business Unit (DP) in CSIRO
  • Commonwealth Scientific and Industrial Research Organisation: federal government agency for scientific research (founded in 1926) with 6600+ people
• Merged into a single CSIRO entity – Data61
  • 1000 + people
  • Innovation network: partnership model
  • Mission: Creating our data-driven future
  • 4 Research Programs: Data Analytics, Decision Science, Software and Computational Systems, Autonomous and Network Systems
Agenda:

• DevOps Overview
• Microservice Architecture
• Continuous Deployment
• Case Study: Implementing a Continuous Deployment Pipeline for Enterprises
• Outlook
Why DevOps?

- Developers (Devs) and operators (Ops) don’t always pursue the same goals

![Meme Image](http://memegenerator.net)

**WORKED FINE IN DEV**

**OPS PROBLEM NOW**
What problem is DevOps trying to solve?

• Poor communication between Dev and Ops
• Opposing goals
  • Devs want to push new features
  • Ops want to keep the system available
    Leads to slow release schedule
• Different cultures
• Limited capacity of operations staff
• Developers have limited insight into operations
Why companies care

• IBM

“IBM has gone from spending about 58% of its development resources on innovation to about 80%”
http://devops.com/blogs/ibms-devops-journey/?utm_content=12855120

• Paddy Power (Ireland):

“The cycle time from a user story's conception to production has decreased from several months to 2 to 5 days.
“Previously, approximately 30% of the workforce was fixing bugs. Now, bugs are so rare that the teams no longer need a bug-tracking system.”

DevOps motivation

• Organizations want to reduce time to market for new features, without sacrificing quality
  • Requires business-IT alignment
• DevOps practices will influence...
  • the way you organize teams
  • the way you build systems
  • even the structure of the systems that you build
• Unlikely that you’ll be able to “throw your final version over the fence” and let operations worry about running it!
DevOps motivation

• DevOps is about...
  • bringing “agile” methods to operations
  • encouraging collaboration between development and operations staff, get them talking
  • Formally: shared goals and teams of Devs and Ops
  • Informally: beer & chips
What is DevOps?

• Definition

“DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality.”

• Source:
  • http://amzn.to/1Qlar8K
  • Release date: June 2015
What does that mean?

• Quality of the code must be high
  • Testing & test-driven development

• Quality of the build & delivery mechanism must be high
  • Automation & more testing
  • A must when deploying to production 25x per day (etsy.com)

• Time is split:
  • From commit until deployment to production
  • From deployment until acceptance into normal production
  • Means testing in production

• Goal-oriented definition
  • May use agile methods, continuous deployment (CD), etc.
  • Likely to use automation tools
DevOps Practices (1/2)

- Treat Ops as first-class citizens throughout the lifecycle – e.g., in requirements elicitation
  - Many decisions can make operating a system harder or easier
  - Logging and monitoring to suit Ops
- Make Dev more responsible for relevant incident handling
  - Shorten the time between finding and repairing errors
DevOps Practices (2/2)

• Use continuous deployment, automate everything
  • Commits trigger automatic build, testing, deployment
• Enforce deployment process is used by all
  • No ad-hoc deployments
  • Ensures changes are traceable
• Develop infrastructure code with the same set of practices as application code
  • “Infrastructure as Code” : using IaaS APIs, etc., to automate creation of environments
  • Misconfiguration can derail your application
  • Ops scripts are traditionally more ad-hoc
Agenda:

- DevOps Overview
- Microservice Architecture
- Continuous Deployment
- Case Study: Implementing a Continuous Deployment Pipeline for Enterprises
- Outlook
DevOps consequences

Architecturally significant requirement:

*Speed up deployment through minimizing synchronous coordination among development teams.*

- Synchronous coordination, like a meeting, adds time since it requires
  - Ensuring that all parties are available
  - Ensuring that all parties have the background to make the coordination productive
  - Following up to decisions made during the meeting
DevOps consequences

• Keep teams relatively small
  • Amazon’s “two pizza rule”: no team should be larger than can be fed with two pizzas
  • Advantages: make decisions quickly, less coordination overhead, more coherent units

• Team size becomes a major driver of the overall architecture:
  • Small teams develop small services → Microservices
  • Coordination overhead is minimized by channeling most interaction through service interfaces:
    – Team X provides service A, which is used by teams Y and Z
    – If changes are needed, they are communicated, implemented, and added to the interface.
Microservice Architecture

• Used in practice by organizations that adopted (or invented) many DevOps practices
  • Amazon, LinkedIn, Google, ...
• Each service provides small amount of functionality
• Total system functionality comes from composing multiple services
• Support different upgrade strategies
  • Rolling and Blue/Green
• Support quick rollback ..
• Design for Failure
Microservice Architecture

• Each user request is satisfied by some sequence of services
• Most services are not externally available
• Each service communicates with other services through service interfaces
• Service depth may be 70, e.g., LinkedIn
Amazon design rules

- All teams will henceforth expose their data and functionality through service interfaces.
- Teams must communicate with each other through these interfaces.
- There will be no other form of inter-process communication allowed:
  - no direct linking, no direct reads of another team’s data store,
  - no shared-memory model, no back-doors whatsoever.
  - The only communication allowed is via service interface calls over the network.
- It doesn’t matter what technology they[services] use.
- All service interfaces, without exception, must be designed from the ground up to be externalizable.
Another Related emerging Architecture
Monitoring/Log-Centric
Agenda:

• DevOps Overview
• Microservice Architecture
• Continuous Deployment
• Case Study: Implementing a Continuous Deployment Pipeline for Enterprises
• Outlook
Continuous Deployment Pipeline

- Developer wants to commit code
- Pre-commit tests are executed locally. If successful:
  - Code is committed
  - Committed code is compiled, Unit tests are run. If successful:
    - Code is built & packaged
      - Result can be a machine image or template (assuming virtualization). If successful:
    - Integration tests are run. If successful:
    - Acceptance / performance tests are run. If successful:
    - The new service is deployed to production

All gates are automatic – else continuous integration / delivery
Deployment is not trivial

Challenges

- 24/7 availability is base requirement
- Microservice 3 to be replaced with new version
  - Multiple VMs for it
- Might change how M1 / M2 can use it
- Might change how it uses M4 / M5
Deployment goal and constraints

• Goal: move from current state (\(N\) VMs of version \(A\) of a service) to new state (\(N\) VMs of version \(B\) of a service)

• Constraints:
  • Any development team can deploy their service at any time – no synchronization among development teams
  • It takes time to replace one VM of version \(A\) with an VM of version \(B\) (order of minutes)
  • Service to clients must be maintained while the new version is being deployed (24/7 availability)
Deployment strategies

• Two basic all-or-nothing strategies:
  • Big Flip (or Blue/Green) Deployment
    – leave $N$ VMs with version $A$ as they are, allocate and provision $N$ VMs with version $B$, switch to version $B$; once stable, release VMs with version $A$.
  • Rolling Upgrade
    – allocate a new VM with version $B$, release one VM with version $A$. Repeat $N$ times.

• Partial strategies
  • Canary testing
  • A/B testing
  • ...
Blue/Green deployment (1/3)

Stage 0
v 1.0 running

Users

my-app.com

Load Balancer

VM

v 1.0

Database

26
Blue/Green deployment (2/3)

Stage 0
v 1.0 running

Users
my-app.com
Load Balancer
VM
v 1.0
Database

Stage 1
v 2.0 infrastructure created and tested

Users
my-app.com
Load Balancer
VM
v 1.0
Database

Testers
my-app.com
Load Balancer
VM
v 2.0
Blue/Green deployment (3/3)

Stage 0
v 1.0 running

Stage 1
v 2.0 infrastructure created and tested

Stage 2
DNS Alias reassigned and users migrated

Users → my-app.com → Load Balancer → VM (v 1.0) → Database

Users → my-app.com → Load Balancer → VM (v 1.0) → Database

Users → my-app.com → Load Balancer → VM (v 1.0) → Database

Users → my-app.com → Load Balancer → VM (v 1.0) → Database
Rolling Upgrade
Rolling Upgrade

Client

Load Balancer

VM v. A
VM v. A
VM v. A
VM v. B booting

Next service
Rolling Upgrade

Client

Load Balancer

VM v. A
VM v. A
VM v. A
VM v. B

Next service
Rolling Upgrade

Client

Load Balancer

VM v. A

VM v. A

VM v. B

Next service
Rolling Upgrade

Client

Load Balancer

VM v. A

VM v. A

VM v. B

VM v. B

Next service
Rolling Upgrade

Client

Load Balancer

VM v. A

VM v. B

VM v. B

Next service
Rolling Upgrade
Rolling Upgrade

Client

Load Balancer

VM v. B
VM v. B
VM v. B

Next service
Blue/Green vs. Rolling Upgrade

Blue/Green Deployment
• Only one version available to the client at any time
• Requires $2N$ VMs
  • additional cost
• Rollback is easy

Rolling Upgrade
• Multiple versions are available for service at the same time
• Requires $N+1$ VMs
  • Can be done at nearly no extra cost
Live testing: Canary testing

• Named after canaries in coal mines
• Small number of servers of a new version, running in production in order to perform live testing in the real environment
Implementation of canaries

• Create set of new VMs as canaries (they are unaware of that)

• Designate a collection of customers as testing the canaries. Can be, for example
  • organization-based (dog-fooding)
  • geographically based
  • at random

• Then
  • Route messages from canary customers to canaries
    – Can be done through making registry/load balancer canary-aware
  • Observe the canaries closely
  • Decide on rolling out / back
Other live testing topics

- A/B testing
  - Have two versions, A and B, run in parallel in production
  - Observe the reactions of users to them
    - E.g., more ad revenue with version B

- Netflix Simian Army
  - Chaos Monkey: randomly kills VMs in production
  - Latency Monkey
  - ...


Agenda:

• DevOps Overview
• Microservice Architecture
• Continuous Deployment
• Case Study: Implementing a Continuous Deployment Pipeline for Enterprises
• Outlook
Case study background

• Case study chapter: “Implementing a Continuous Deployment Pipeline for Enterprises” with John Painter and Daniel Hand

• “Sourced Group is an enterprise consulting organization, working on bringing the benefits of Cloud-based solution architecture and automation to the enterprise.” – http://www.sourcedgroup.com.au/
CDP tooling (in the case study)

- Ticketing integration: tracing of change reasons – can even link release notes to original feature request
- Bamboo: unlike many alternatives has good support for branches
Core feature of CDP: standardized life cycle

- Define life cycle plan globally
  - Tasks: see next slides
- Run virtual plans for each branch – avoids plan drift
  - Branch awareness supported by Bamboo
Standardized life cycle (1/3)

• Building and testing:
  • functional testing and producing application artifacts

• Baking:
  • Bootstrapping application artifacts and configuration onto a temporary target OS
  • “baking” into image: snapshot from which new VMs can be created. AWS: Amazon Machine Image (AMI)
Standardized life cycle (2/3)

- **Deployment:**
  - new, independent stack of the application with new VMs (from the AMI) in an autoscaling group (ASG)
  - supporting infrastructure and configuration: load balancing, scaling, monitoring, networking

- **Release:**
  - Warm up new stack
  - Release by changing the DNS entry
  - Optional: patching or modification of persistent data
Standardized life cycle (3/3)

• (In use – no CDP involvement)
• Teardown:
  • Once no longer needed, stack can be torn down
  • Ensure all traffic has been moved away from stack
  • Remove all resources: VMs, ASG, supporting infrastructure, ...
Environment Definition: AWS CloudFormation

• CloudFormation (CF):
  • Declarative configuration – i.e., what is the goal state
  • Define a complete virtual environment, including resources and security components
  • JSON file – example:

```json
"Resources" : {
  "Route53DNSRecord" : {
    "Type" : "AWS::Route53::RecordSet",
    "Properties" : {
      "HostedZoneName" : mydomain.com.,
      "Comment" : "Application DNS Record",
      "Name" : "myapplication.",
      "Type" : "CNAME",
      "TTL" : "10",
      "ResourceRecords" : "myapplication-",{"Ref: "BuildNumber"},".mydomain.com"
    }
  }
}
```
Layering CloudFormation scripts

- Application code
- Application CF
- Load Balancing setup

- VPC
- Subnets
- Security groups
- CF Frameworks
- Best Practice

Application Repo (Git) → Atlassian Bamboo → CloudFormation (CF) script

Operations Repo (Git)
Case study details: baking

Checkout CF template & build scripts
Merge application & operational CF templates
Upload CF templates to Amazon S3
Pre-Processing

Build stack from CF templates
Bake AMIs from EC2 instances (one per tier)
Post-Processing
Teardown EC2 instances
Agenda:

• DevOps Overview
• Microservice Architecture
• Continuous Deployment
• Case Study: Implementing a Continuous Deployment Pipeline for Enterprises
• Outlook
Motivation: System Operations Matter

• Gartner predicts:
  • “Through 2015, **80% of outages** impacting mission-critical services will be caused by people and process issues, and more than 50% of those outages will be caused by **change / configuration / release integration** and hand-off issues.”

• The case of Knight Capital – from Wikipedia:
  • The Knight Capital Group was an American global financial services firm [...].[...] Knight was the largest trader in U.S. equities, with a market share of 17.3% on NYSE and 16.9% on NASDAQ.[2] The company agreed to be acquired by Getco LLC in Dec 2012 after an trading error lost $460 million.
  • This took 45 minutes and was an upgrade error
Some future DevOps topics

• Agility for all!
  • Business initiatives take long to implement – move towards “BizOps”
  • Data scientists – make operation of complex bespoke solutions smoother, more flexible, and more robust
  • Security & compliance – easier and faster changes

• Managing Microservices
  • Co-owned test suites to ensure upstream services’ (future) requirements are met
  • How much power to give each team, vs. centralized process and policy enforcement?
  • How to manage mature microservices, e.g., after re-org?
Summary

• DevOps
  • Fast deployment of new features to production while ensuring high quality

• Microservices & Architecture
  • Split application into small, well-scoped components

• Continuous Deployment
  • Deployment without downtime, e.g. Blue/Green, Rolling Upgrade

• Case study
  • CDP for enterprises

• Many more topics
  • 3 case studies in the book show how things can be done in practice
DevOps: A Software Architect’s Perspective

• Research Partnership with Data61 and CSIRO
  • 1000+ Scientists on ICT/Software/Data related technologies
    – DevOps, SE but many more
• Data61+ Program
• CSIRO China Office

Q & A?

Contact:
Liming.Zhu@data61.csiro.au

http://amzn.to/1Qlar8K
Release date: June 2015