Carnegie Mellon

Why architects should care about DevOps

Len Bass

What is DevOps?

- "[DevOps] aims to shorten the systems development life cycle and provide continuous deployment with high software quality"
- DevOps is a process improvement effort. This means there must be measurements
  - deployment time and frequency
  - production tickets and time to repair

DevOps differs from prior process improvement efforts

DevOps Processes

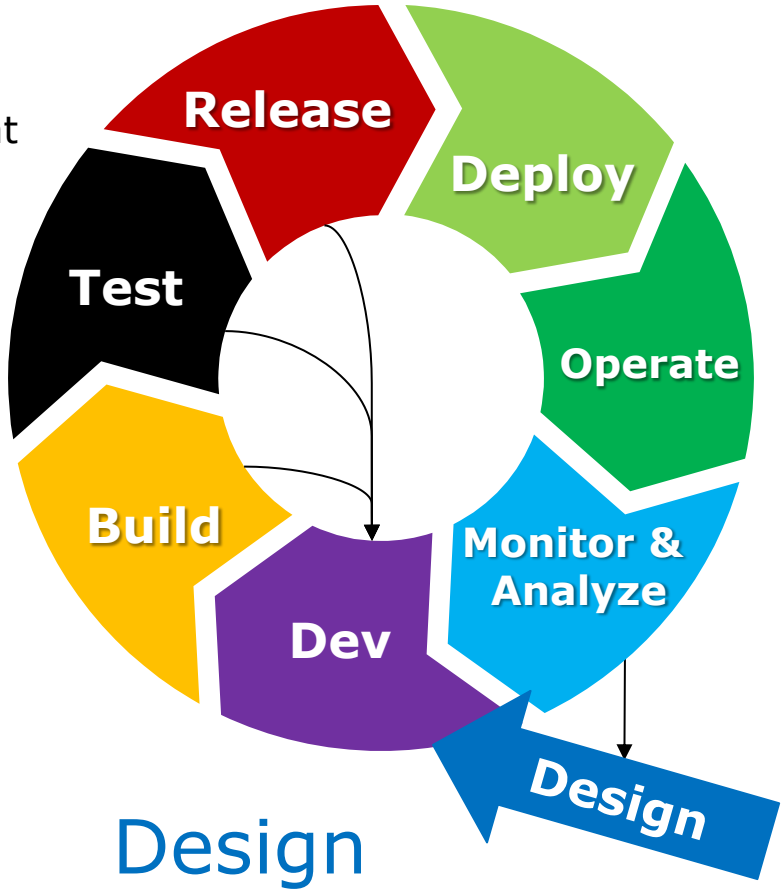
Release
Approve for deployment

Test
Ensure high test coverage & automate tests as much as possible

Build
Create an executable artifact

Dev
Perform normal development activities

Create scripts for other activities



Design
Design architecture to support other activities

Deploy
Move into production environment

Operate
Execute system and gather measurements about its operation

Monitor & Analyze
Display measurements taken during operation & analyze the data

Carnegie Mellon University

Managing Credentials

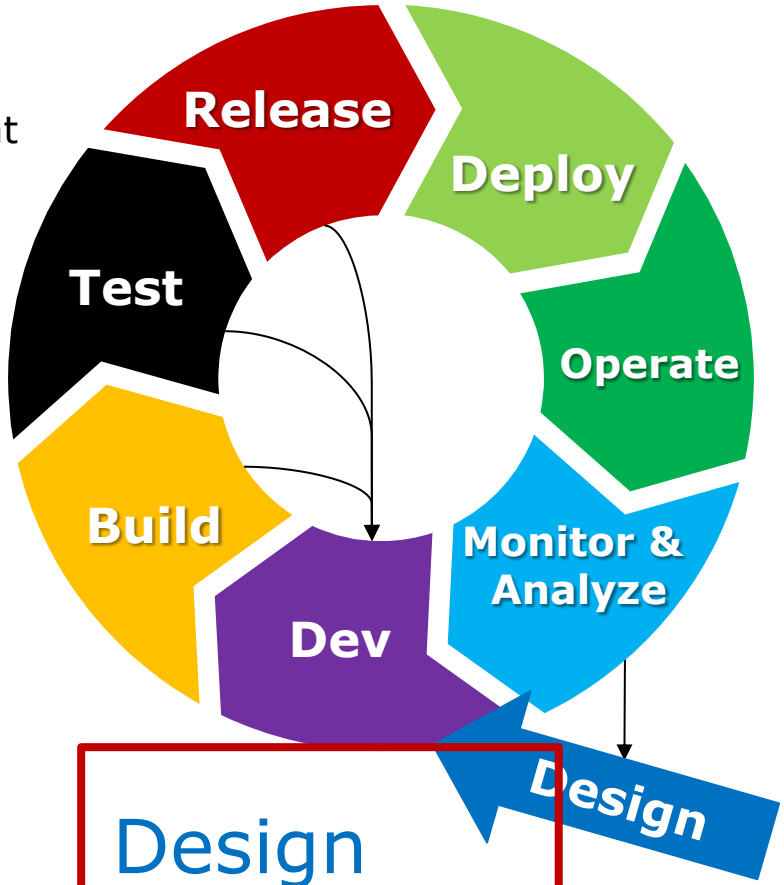
Release
Approve for deployment

Test
Ensure high test coverage & automate tests as much as possible

Build
Create an executable artifact

Dev
Perform normal development activities

Create scripts for other activities



Design
Design architecture to support other activities

Deploy
Move into production environment

Operate
Execute system and gather measurements about its operation

Monitor & Analyze
Display measurements taken during operation & analyze the data

isr institute for SOFTWARE RESEARCH

Credential Sprawl

- Allowing applications to manage their own credentials leads to *credential sprawl*.
  - Difficult to determine where credentials exist
  - Difficult to prevent leakage of credentials
  - Difficult to determine audit trail
  - Difficult to rotate credentials

Vault

- Vault is a centralized credential management system
- It manages
  - Credentials for access to resources
  - Audit trail of access
  - Credential rotation

Use of Vault

- Applications must be designed to use Vault for storing and retrieving credentials.
- Resource managers are registered with Vault.
- Vault Is integrated with OAuth to allow an application or user access to a resource

Carnegie Mellon University

Managing Scripts

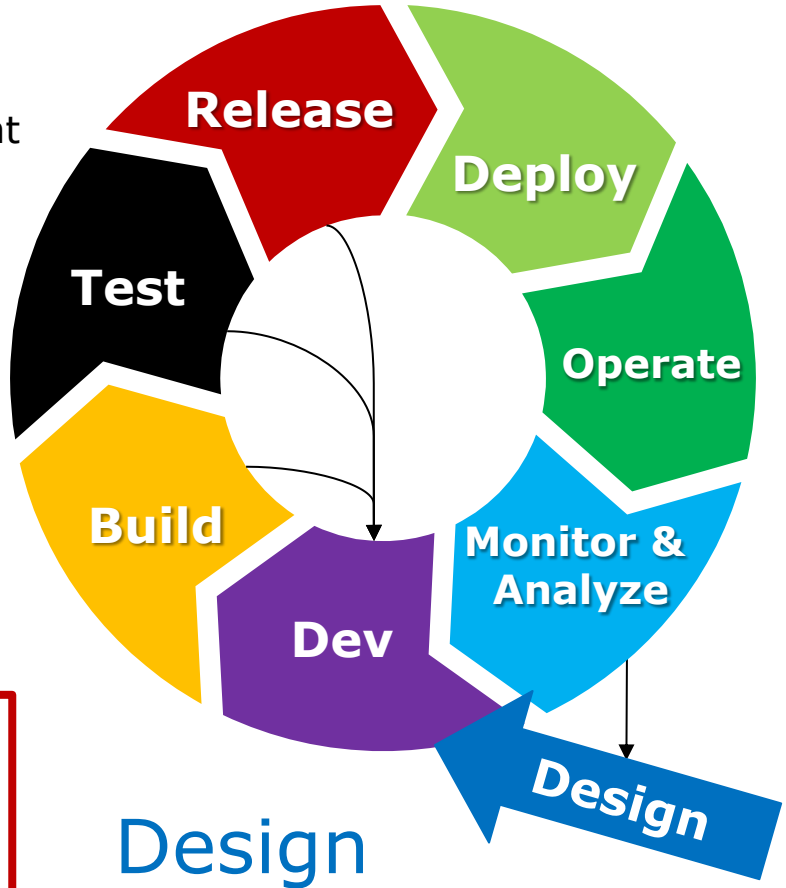
Release
Approve for deployment

Test
Ensure high test coverage & automate tests as much as possible

Build
Create an executable artifact

Dev
Perform normal development activities

Create scripts for other activities



Design
Design architecture to support other activities

Deploy
Move into production environment

Operate
Execute system and gather measurements about its operation

Monitor & Analyze
Display measurements taken during operation & analyze the data

What is IaC?

- Infrastructure as Code (IaC) is the management of infrastructure (networks, virtual machines, load balancers, and connection topology) using code segments in various languages. E.g.
  - Command line scripting
  - Provisioning specifications, e.g. Vagrant, Terraform, Cloud Formation, Chef, Puppet, Ansible
  - Specification files for various tools, e.g Dockerfile
- Every repetitive action should be automated.

Management of IaC

- IaC is managed in the same fashion as code
  - Version controlled
  - Shared among teams
  - Tested

IaC languages

- IaC languages are typically declarative although they may have some imperative portions.

Deployment Architecture

- Includes VMs, network connections, security settings,
- Must be constructed correctly
- Must be able to be reconstructed
- Specifying deployment architecture as IaC supports these requirements

Cloud providers provide templates

- The constrction of an IaC program is simplified by the availability of templates provided by cloud providers.

Sample Cloud Formation template (with no access control)

```
"AWSTemplateFormatVersion" : "2010-09-09",
"Description" : "A sample template", "Resources" : {
"MyEC2Instance" : { "Type" : "AWS::EC2::Instance",
"Properties" : { "ImageId" : "ami-2f726546",
"InstanceType" : "t1.micro", "KeyName" : "testkey",
"BlockDeviceMappings" : [ { "DeviceName" :
"/dev/sdm", "Ebs" : { "VolumeType" : "io1", "Iops" :
"200", "DeleteOnTermination" : "false", "VolumeSize" :
"20" } } ] } } } }
```

Vulnerabilities

- However, inappropriate access control is a source of many production vulnerabilities.
- One study by Palo Alto Networks found over 200,000 vulnerabilities in CloudFormation specifications due to inadequate access controls.
- Another study cited by Accurics found that 93% of specifications had misconfigured cloud storage access controls.

Deployment

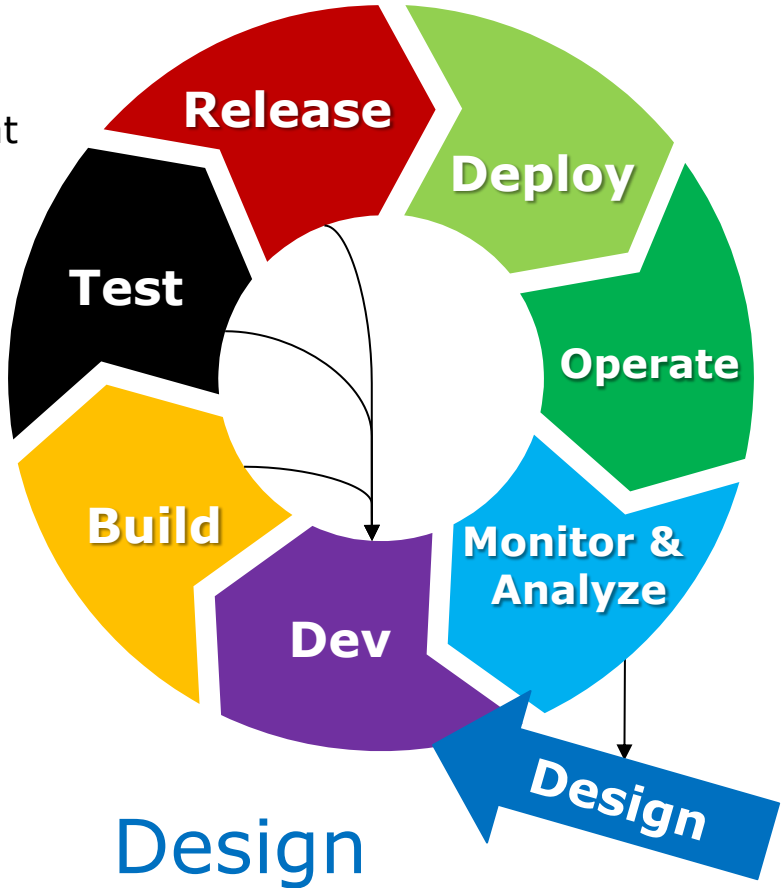
Release
Approve for deployment

Test
Ensure high test coverage & automate tests as much as possible

Build
Create an executable artifact

Dev
Perform normal development activities

Create scripts for other activities



Design
Design architecture to support other activities

Deploy
Move into production environment

Operate
Execute system and gather measurements about its operation

Monitor & Analyze
Display measurements taken during operation & analyze the data

What is continuous delivery/deployment?

- "With continuous delivery, every code change is built, tested, and then pushed to a non-production testing or staging environment.
- The difference between continuous delivery and continuous deployment is the presence of a manual approval to update to production. With continuous deployment, production happens automatically without explicit approval." (Amazon)

Continuous deployment

- When a team completes revisions on their service
  - They commit it to a version control system
  - This triggers the deployment pipeline
  - If no errors are discovered, it goes directly into production
- No interruption of service.

Any team can deploy at any time

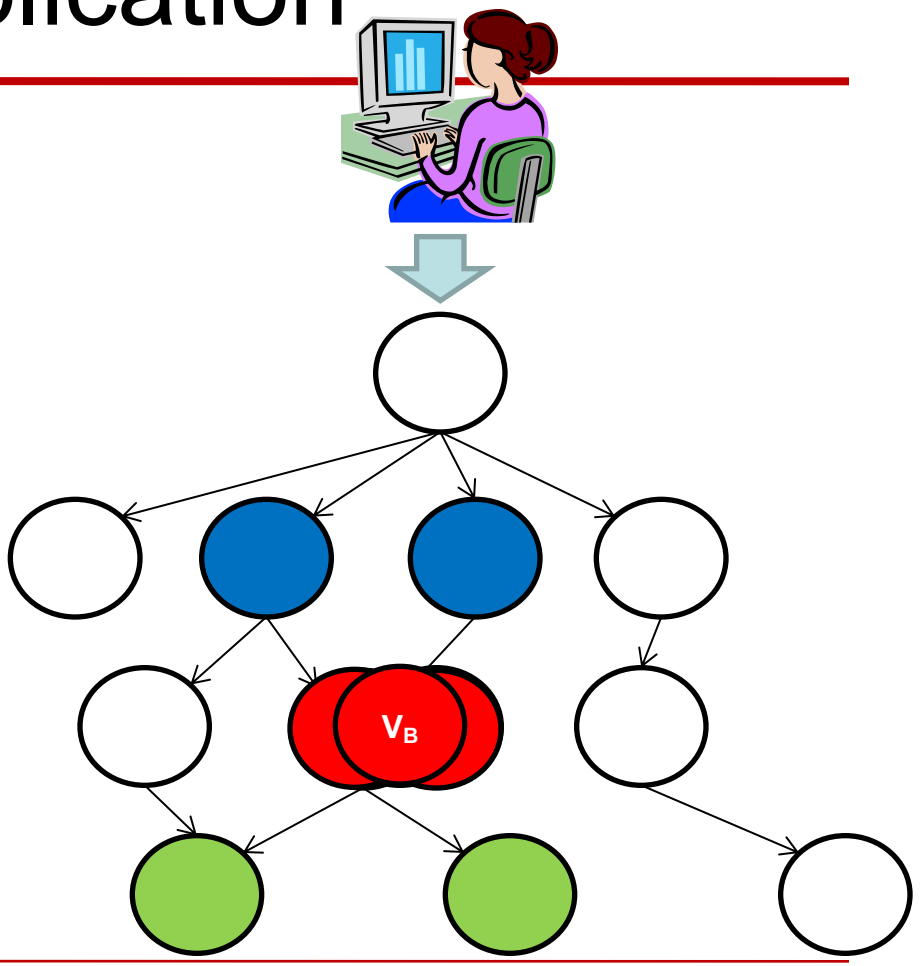
- In traditional release cycles, teams coordinate so that modifications to services are placed into production simultaneously.
- With continuous deployment the situation is different
  - Any team can deploy at any time
  - There is no coordination among teams with respect to sequencing of service deployments.

Deploying a new version of an application

Multiple instances of a service are executing

- Red is service being replaced with new version
- Blue are clients
- Green are dependent services

Staging/container repository



Deployment goal and constraints

- Goal of a deployment is to move from current state (N instances of version A of a service) to a new state (N instances of version B of that service)
- Constraints:
  - Any development team can deploy their service at any time. I.e. New version of a service can be deployed either before or after a new version of a client. (no synchronization among development teams)
  - It takes time to replace one instance of version A with an instance of version B (order of minutes for VMs)
  - Service to clients must be maintained while the new version is being deployed.

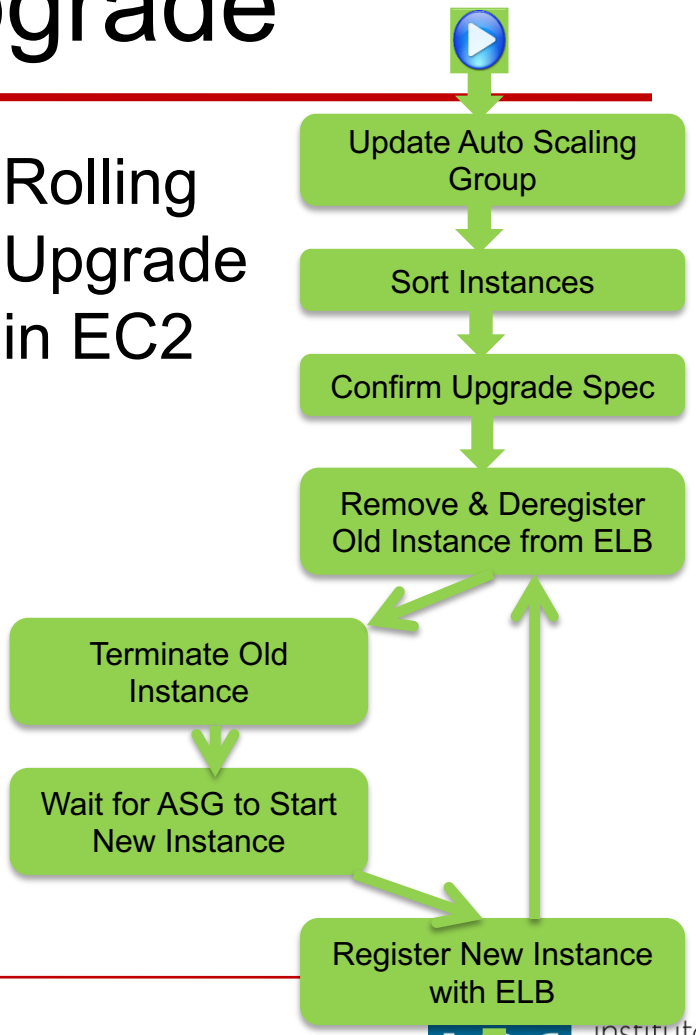
Deployment strategies

- Two basic all of nothing strategies
  - Red/Black (also called Blue/Green) – leave N instances with version A as they are, allocate and provision N instances with version B and then switch to version B and release instances with version A.
  - Rolling Upgrade – allocate one instance, provision it with version B, release one version A instance. Repeat N times.
- Partial strategies are canary testing and A/B testing.

Trade offs – Red/Black and Rolling Upgrade

- Red/Black
  - Only one version available to the client at any particular time.
  - Requires 2N instances (additional costs)
- Rolling Upgrade
  - Multiple versions are available for service at the same time
  - Requires N+1 instances.
- Rolling upgrade is widely used.

Rolling Upgrade in EC2



Version skew

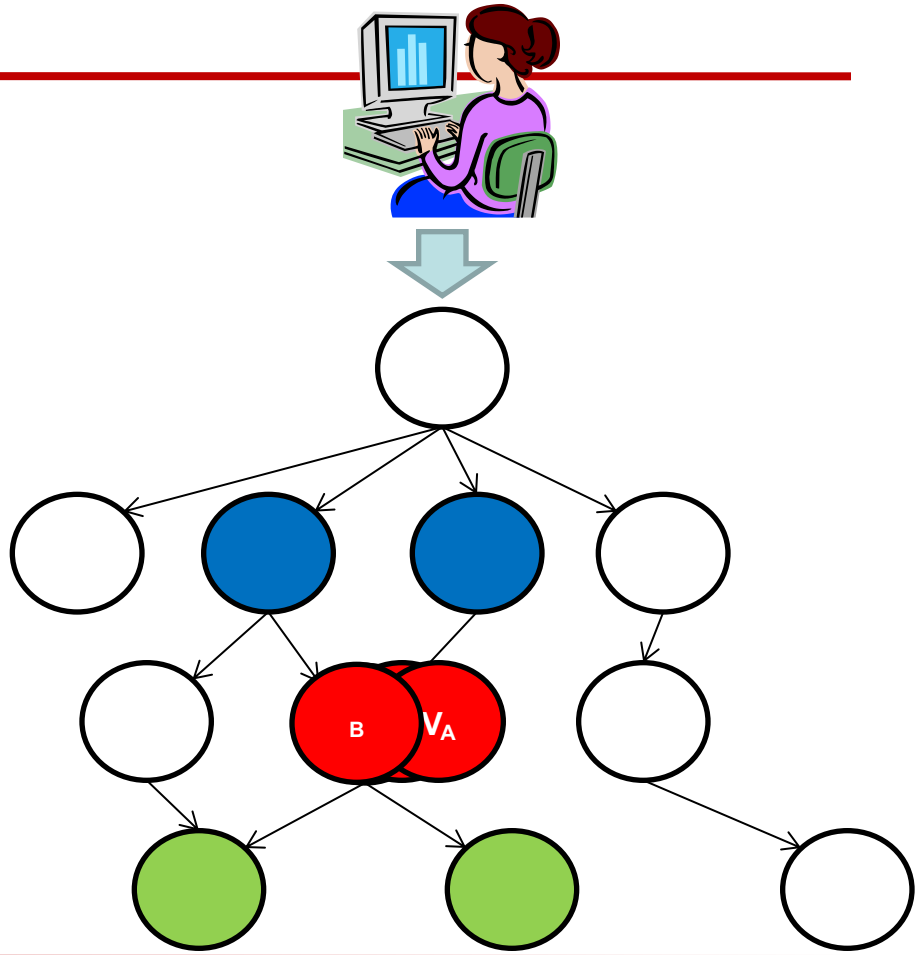
- Teams can deploy new version of a service without coordinating with other teams. Other teams are managing clients and dependent services.
- This leads to possible inconsistencies among versions
  - Client may have been updated to new version whereas server has not
  - Vice versa
- We call this type of version skew “temporal inconsistency”

Version skew example

- Suppose service A calculates the price of an item on a shopping cart – including discount.
- The organization is changing the model for discounting.
  - Previously discount was per item
  - Now the discount is based on total purchases.

Version skew example

- Blue service enumerates items in the shopping cart by invoking services.
- New discount model requires Blue service to calculate discount.
- If Red service has been updated and blue has not, no discounts are calculated.
- Vice versa yields two discounts.



Interface mismatch

- Another type of version skew occurs if an interface is modified.
- In this case, the recipient should maintain backward or forward compatibility.
- Backward compatibility means calling an old interface still works correctly.
- Forward compatibility means the recipient recognizes incorrect interface and responds appropriately.

Managing version skew

- Version interfaces.
- Any modification to a service should result in a new version number for its interface.
- Tag messages with version number of expected interface
- It becomes the responsibility of the invoked service to manage messages expecting different versions

Protocol Buffers

- A protocol buffer specification is used to specify an interface. Kept in a .proto file
- Language specific compilers used for each side of an interface
- Allows different languages to communicate across a message based interface
- Collection of .proto files defines all of the interfaces and hence all of the services.

Protocol Buffers/Version Skew

- Protocol buffer specification is kept in a version control system. I.e, It has a version number.
- The compiler can include tagging the message with the version number of the .proto file in the generated code.
- If you use protocol buffers, you can manage version skew by modifying recipient service.

Operation

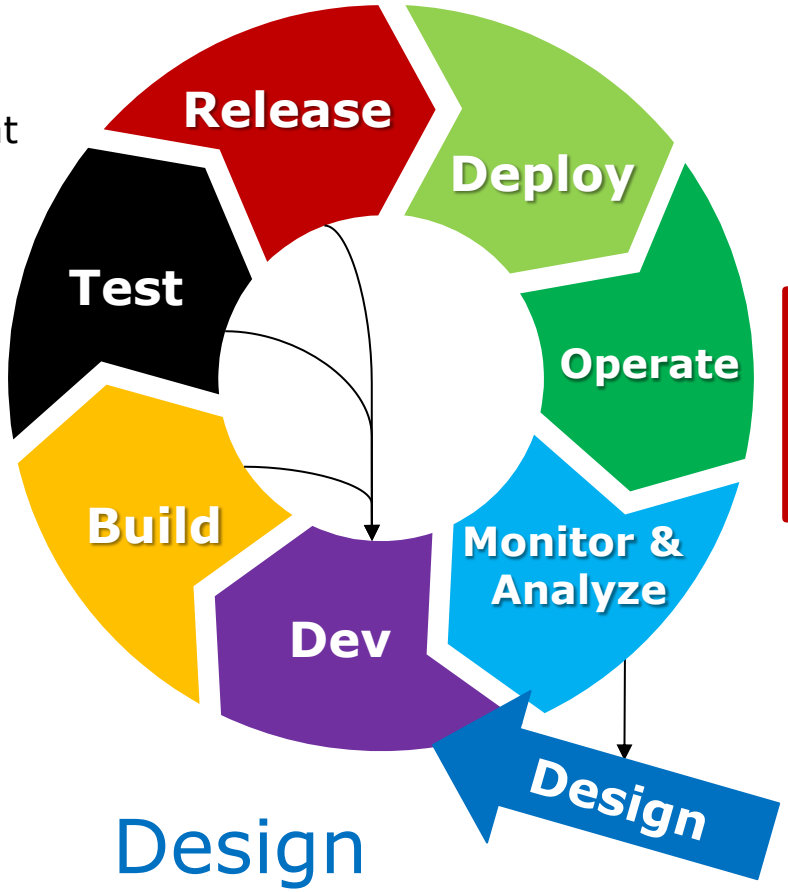
Release
Approve for deployment

Test
Ensure high test coverage & automate tests as much as possible

Build
Create an executable artifact

Dev
Perform normal development activities

Create scripts for other activities



Design
Design architecture to support other activities

Deploy
Move into production environment

Operate
Execute system and gather measurements about its operation

Monitor & Analyze
Display measurements taken during operation & analyze the data

Chaos Engineering

- Chaos Engineering is the discipline of experimenting on a system in order to build confidence in the system's capability to withstand turbulent conditions in production.
- *Experiment* and *in production* are key.
- Assumption is that testing large distributed systems in the face of disruptions is not possible.

Steps 1,2 for experiment

1. Define ‘steady state’ as some measurable output of a system that indicates normal behavior.
2. Hypothesize that this steady state will continue in both a control group and a experimental group.

Steps 3,4

3. Introduce variables that reflect real world events like servers that crash, hard drives that malfunction, network connections that are severed, etc.
4. Try to disprove the hypothesis by looking for a difference in steady state between the control group and the experimental group.

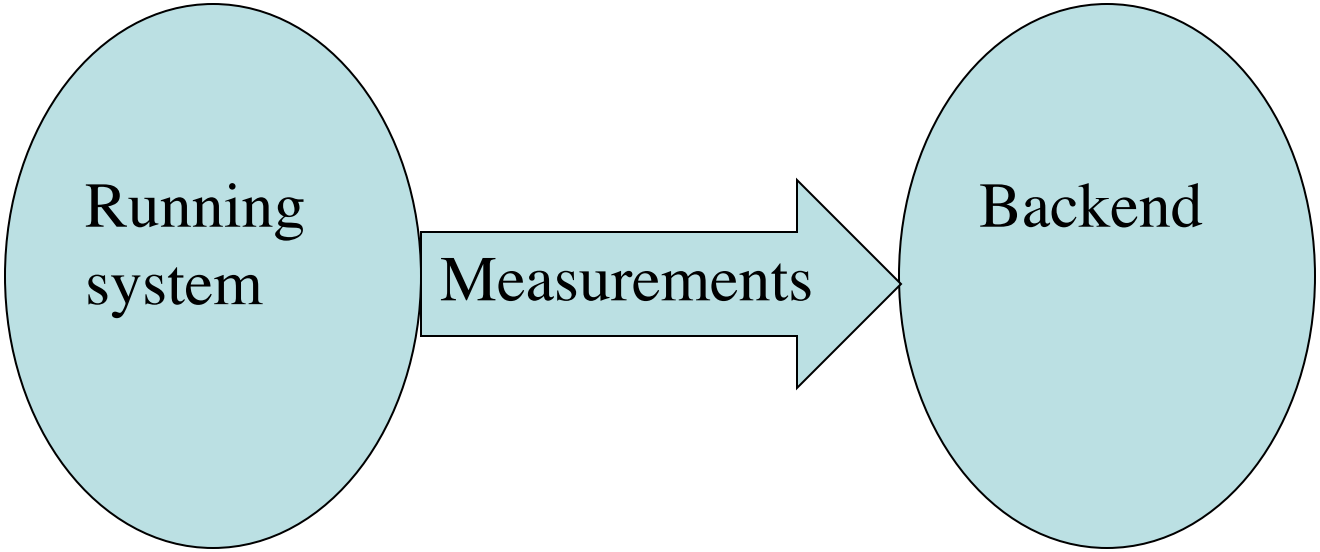
Chaos Monkey

- Classic example is the Chaos Monkey – Netflix and Google.
- It randomly kills servers in production.
- Effect should not be observable by end user.

Measurement

- After a system is in operation, measurement information is gathered for three purposes:
  - Alerting – Detecting that there is a problem
  - Forensics – determining what caused a problem
  - Improvement – finding bottlenecks in systems or determining causes of internet traffic.

General picture



Measurements are taken from a running system and its environment and sent to a back end.
Splunk is common backend system.

Back end

- The back end has a data base (usually a time series database). It
  - Generates alerts
  - Generates reports
  - Allows drilling down into aggregate information to get more detailed information
  - Has a dashboard to give fast indication of problems.

Alerts

- Back end has set of rules to establish when to send an alert. E.g. alert if CPU utilization is over 80% for 15 minutes.
- Utilization numbers are bursty. The period must be sufficiently long to indicate problem.
- False positives and false negatives are both problems.
- An alert causes a page to be sent.

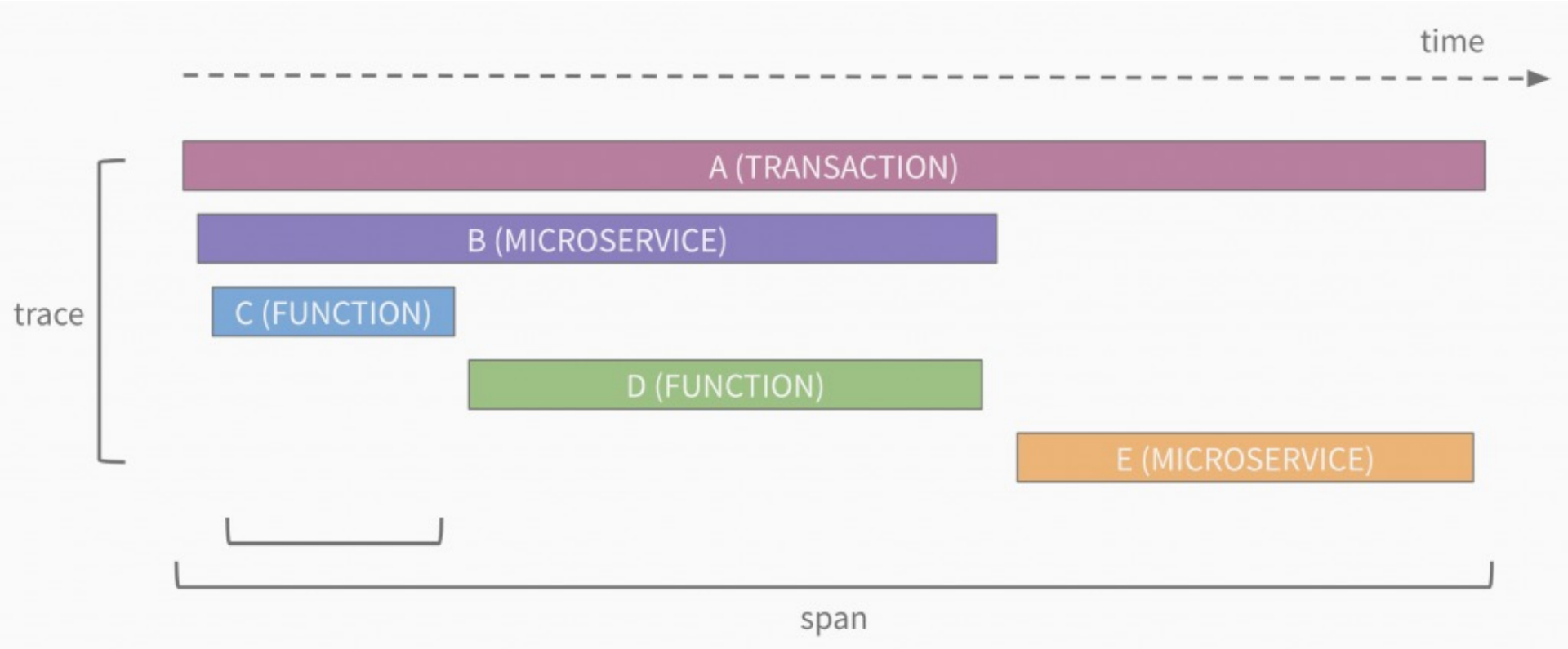
Thresholds for Alerts

- SLA – Service Level Agreement. What is guaranteed to clients (internal or external) for each indcator
- SLO – Service Level Objective. A goal for the team for each agreement. More restrictive than SLA
- SLI – Service Level Indicator. Measurement of the objective. For each indicator, define an SLI and alert when it is violated.

Spans

- A trace captures the end to end actions in response to a user request.
- A span is a named, timed operation that represents a piece of the trace.
- Spans may have child spans
- Displaying spans on a time axis allows you to see:
  - Parallelism
  - Where time is being spent

Sample span display*



*https://sflanders.net/2019/03/28/an-intro-to-distributed-tracing/

How does tracing work? - 1

- This is one of many possible implementations.
- Request enters the system from external source – user or external system
- Request is given a unique ID that reflects the context
- Context description is kept in a data base so that with context ID analyst can know details of the context.

How does tracing work? – 2

- Context id becomes a portion of HTTP header. World Wide Web Consortium is standardizing how this will work.
- The context ID is inserted by the HTTP server accepting the request and propagated by every service as it fans out the request. This is transparent to the requester.
- Context can be used to control behavior of a service.

Examples of context

- Test version. Use context to affect behavior or routing.
- Application. Google might want to know what percentage of their network traffic might be due to search, Gmail, etc
- Traffic prioritization. Give priority to certain requests to maintain quality of service.
- Bottleneck determination. Where is the most time being spent in a collection of transactions?

Site Reliability Engineer (SRE)

- An SRE is first responder when an alert occurs.
  - Their responsibility is to determine immediate cause of problem
  - Get system back into operation
  - Determine underlying problem cause
  - inform development team of cause

SRE skill set

- Overall view of how system fits into infrastructure
  - Individual components
  - Interactions with infrastructure
- Good problem-solving skills
- Good communication skills
- SREs are software architects with a different title.

Future – tool evolution

- Vendors will consolidate. Tools will get merged.
- Tools will expand to cover more of the DevOps processes
- Patch management software market expected to double in next 2 years.

Future – multi-cloud

- The multi-cloud market is projected to grow to ~$30 billion by 2028.
- Multi-cloud improves reliability and avoids vendor lock in.
- Has the cost of ensuring all systems,tools, and IaCs run on both vendors.

Avoiding vendor lock in for cloud providers

- Some provisioning tools are cloud provider agnostic, others are cloud provider specific.
- The trade off is traditional. Agnostic tools can be used on multiple vendors whereas specific tools can take advantage of cloud vendor specific features.

Avoiding vendor lock in for tools

- It is possible to get locked in to tool vendors as well as cloud providers
- Expect to see translation mechanisms from one tool vendor's language to another.
- Many popular DevOps tools are open source: Docker, Kubernetes, Jenkins, Vault, Istio, Ansible, Chef, Terraform

Future – domain specific

- Expect to see growth of domain specific DevOps
  - DevOps for AI. Separate pipeline for data and software.
  - DevOps for government. DevOps practices and tools included in RFPs/contracts.
  - DB DevOps. Management of database code changes

Summary

- DevOps is a process improvement effort concerned with deployment time and incident handling time.
- Software architects must design for
  - version skew
  - Measurement
  - Centralized credential management
- Chaos Engineering is a discipline of testing in production

Summary

- Multi-cloud hosting will grow
- Domain specific DevOps will grow
- Tool vendors will consolidate
- Vendor lock in will become a bigger problem.

More Information

- "Deployment and Operations for Software Engineers" is available from Amazon.