Lecture 7 - Devops

History

- New invention in the age of web 2.0 and internet
- In the olden days (as recent as early 2000s), software are delivered on physical media, often w/o even a way to automatically update, and certainly no need for servers.
- As a result of this process, you are doing a public release maybe once a year, or even once every multiple years in the case of a large software like operating system
- Testing was mostly done manually, and you have regular build cuts that QA testers install on their computer and then actually running through manual tests

2000s - 2010

- Software as a Service, SaaS, popularized by Salesforce and Google (gmail/google maps), and the advent of Ajax and not requiring a reload every time someone needs new data
- Typically deployed to owned data centers, or colocated servers in someone else's data centers
- Testing is starting to be automated, and unit tests are popularized here. Still majority QA teams for tests.

2010s - 2020s

- Managed virtual services are created and popularized, especially led by the creation of AWS.
- Hardware became far more elastic, and it became possible to easy to scale up and down services as demand rises and drops.
- At this point, devops became popular

What's devops?

- Development + operations. Back in the early 2000s, most sys admins and folks who are involved in operations are very specialized. They are experts in the systems that they are responsible for, but not necessarily expert coders and aren't necessarily inclined to automate their work as much as software engineers may be.
- As well due to the prevalence of cloud services, a lot of the drudgery where you are requiring a dedicated team of sys admins or security experts no longer apply. You can get professionally managed commoditized services from an vast array of vendors and allow you to focus on your differentiator.
- What used to require a dedicated team to properly scale and productionize, you can now do it in your bedroom in your underwear.

CI/CD

- Key part of devops, it's the main joining of development team with the operations team. With a properly set up CICD pipeline, the commit you made into master or production branch have a more or less automated way
- Two parts, continuous integration and continuous deployment
- Integration is more on the development side. Where each commit you make is continuously tested along with all the other commits (integrated part). Key part of CI is unit tests/integration tests, and other automated process can trigger along with it, such as automated security screening and preparation for review
- By merging frequently and triggering automatic testing and validation processes, you minimize the possibility of code conflict, even with multiple developers working on the same application. A secondary benefit is that you don't have to wait long for answers and can fix bugs and security issues while it's still fresh.
- Delivery means deployment, aka releasing the product to the public. Typically once all the tests and post commit issues are fixed, the code is automatically pushed to staging environments. The key to understand here is that the staging environment should be pretty much the same as the production environment, just on a smaller scale. Everyone who contributed to the release will all gather in the same environment to test their parts to ensure everything is as expected as a sanity test. Then on a regular basis (could be multiple times a day to once a week), a release cut is deployed to production. Oncall engineers would then monitor the deployment to ensure no negative impact on the metrics.
- A couple more real world wrinkles in this CICD party:
    - Mobile builds are hard, typically you need to get app store approvals and there is a huge long tail of people who don't upgrade their app regularly. So if you are looking to deprecate an API that a small minority of the users are still using. You need to think about it carefully and make sure you are comfortable forcing users to upgrade
    - Feature flags are frequently used to make deployments safer. Deployments are not the safest thing in the world, rollbacks because they are done less frequently, are even more dangerous. Feature flags are often used to
        - Support AB testing, where you can say deliver feature x to 50% of the user only and leave the other 50% of the user on the old version to make sure no metrics degrades
        - Support gradual rollout. Often for large features, you do not want to deploy even to a 50/50 AB test. You want to have a gradual ramp from 1% to 100%, and make sure you are monitoring critical metrics to guide the decision to continue ramping.
        - You can use feature flags to disable features in case there are unintended side effects. Feature flags can also be used to test if disabling something will fix the issue

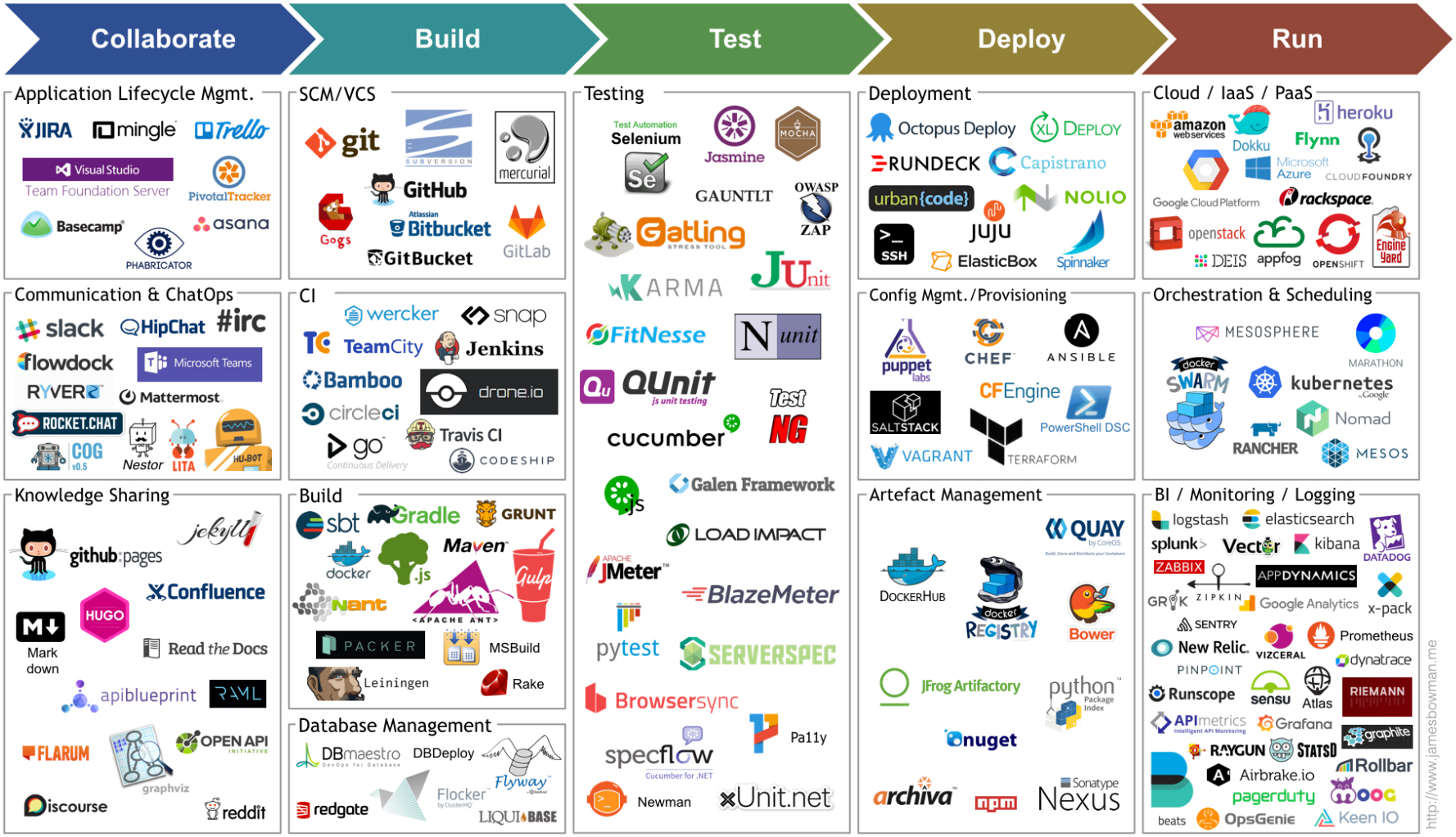
Security & Compliance Tools—
"Sec"Ops"

ITOps or Infra
Architects /
SREs & System /
Administrator(s)

ITOps or Release Managers

Software Developer:
Front-End, Back-End,
Tester QA

Security Compliance or Ops
Security Engineer


Collaborate
Build
Test
Deploy
Run
Application Lifecycle Mgmt.
JIRA
mingle
Trello
Visual Studio
Team Foundation Server
PivotalTracker
Basecamp
asana
PHABRICATOR
SCM/VCS
git
SUBVERSION
mercurial
GitHub
Gogs
Atlassian Bitbucket
GitLab
GitBucket
Communication & ChatOps
slack
HipChat
#irc
flowdock
Microsoft Teams
RYVER
Mattermost
ROCKET.CHAT
COG v0.5
Nestor
LITA
HU-BOT
CI
wercker
snap
TeamCity
Jenkins
Bamboo
drone.io
circleci
Travis CI
go Continuous Delivery
CODESHIP
Knowledge Sharing
github:pages
jekyll
Confluence
HUGO
Mark down
Read the Docs
apiblueprint
RAML
FLARUM
OPEN API INITIATIVE
graphviz
discourse
reddit
Build
sbt
Gradle
GRUNT
docker
Maven
.js
Gulp
Nant
APACHE ANT
PACKER
MSBuild
Leiningen
Rake
Database Management
DBmaestro DevOps for Database
DBDeploy
Flyway
Flocker by ClusterHQ
redgate
LIQUIBASE
Testing
Test Automation
Selenium
Jasmine
MOCHA
GAUNTLT
OWASP ZAP
Gatling STRESS TOOL
JUnit
KARMA
FitNesse
N unit
QUnit js unit testing
Test NG
cucumber
Galen Framework
LOAD IMPACT
APACHE JMeter
BlazeMeter
pytest
SERVERSPEC
Browsersync
Pa11y
specflow Cucumber for .NET
Newman
xUnit.net
Deployment
Octopus Deploy
DEPLOY
RUNDECK
Capistrano
urban{code}
NOLIO
SSH
JUJU
ElasticBox
Spinnaker
Config Mgmt./Provisioning
puppet labs
CHEF
ANSIBLE
SALTSTACK
CFEngine
PowerShell DSC
VAGRANT
TERRAFORM
Artefact Management
QUAY by CoreOS
DOCKERHUB
docker REGISTRY
Bower
JFrog Artifactory
python Package Index
nuget
archiva
npm
Sonatype Nexus
Cloud / IaaS / PaaS
amazon web services
heroku
Dokku
Flynn
Microsoft Azure
CLOUDFOUNDRY
Google Cloud Platform
rackspace
openstack
DEIS
appfog
OPENSHIFT
Engine Yard
Orchestration & Scheduling
MESOSPHERE
MARATHON
docker SWARM
kubernetes by Google
Nomad
RANCHER
MESOS
BI / Monitoring / Logging
logstash
elasticsearch
DATADOG
splunk>
Vector
kibana
ZABBIX
APPDYNAMICS
GROK
ZIPKIN
Google Analytics
x-pack
SENTRY
New Relic.
VIZCERAL
Prometheus
dynatrace
PINPOINT
Runscope
sensu
Atlas
RIEMANN
APImetrics Intelligent API Monitoring
Grafana
graphite
RAYGUN
STATSD
Rollbar
Airbrake.io
pagerduty
BOOC
beats
OpsGenie
Keen IO
http://www.jamesbowman.me