DIVIDE AND CONQUER

EASIER CONTINUOUS DELIVERY USING MICRO-SERVICES

Carlos Sanchez

csanchez.org @csanchez

cloudbees®

AGILE TESTING DAYS ASIA
ASIA's Greatest AGILE Event
ASIA'S GREATEST AGILE SOFTWARE TESTING EVENT!
March 01-03, 2017 in Bengaluru, INDIA

STeP-IN Forum
Software Test Practitioners, INDIA

Watch online at carlossg.github.io/presentations

ABOUT ME

Engineer @ CloudBees, scaling Jenkins

Author/contributor of Jenkins Kubernetes and Mesos plugins

Long time OSS contributor at Apache (Apache Maven), Eclipse, Puppet,…

MICRO SERVICES

---

the microservice architectural style is 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.*

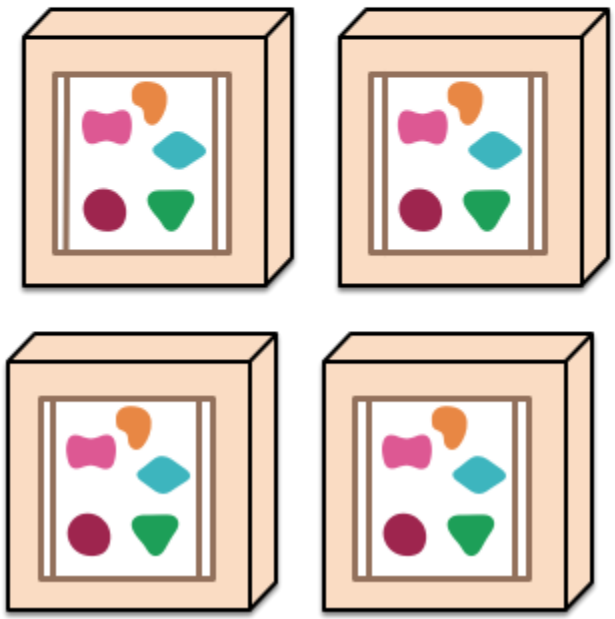
James Lewis and Martin Fowler

- One application, multiple small services
- Separate processes with lightweight comunications, typically HTTP
- Deployed independently
- Minimal centralized management
- Fully automated deployment

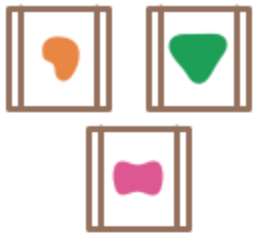
MONOLITH VS MICRO-SERVICES

A monolithic application puts all its functionality into a single process...

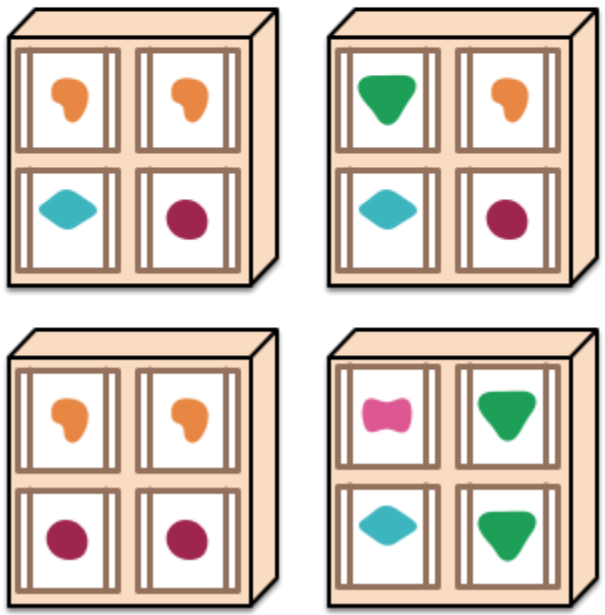
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.



COMPONENTIZATION VIA SERVICES

vs libraries

ORGANIZED AROUND BUSINESS CAPABILITIES

cross functional teams

PRODUCTS NOT PROJECTS

business oriented

ongoing maintenance

DECENTRALIZED GOVERNANCE

different lenguages

Amazon: you build it you run it

DECENTRALIZED DATA MANAGEMENT

each service manages its own database

INFRASTRUCTURE AUTOMATION

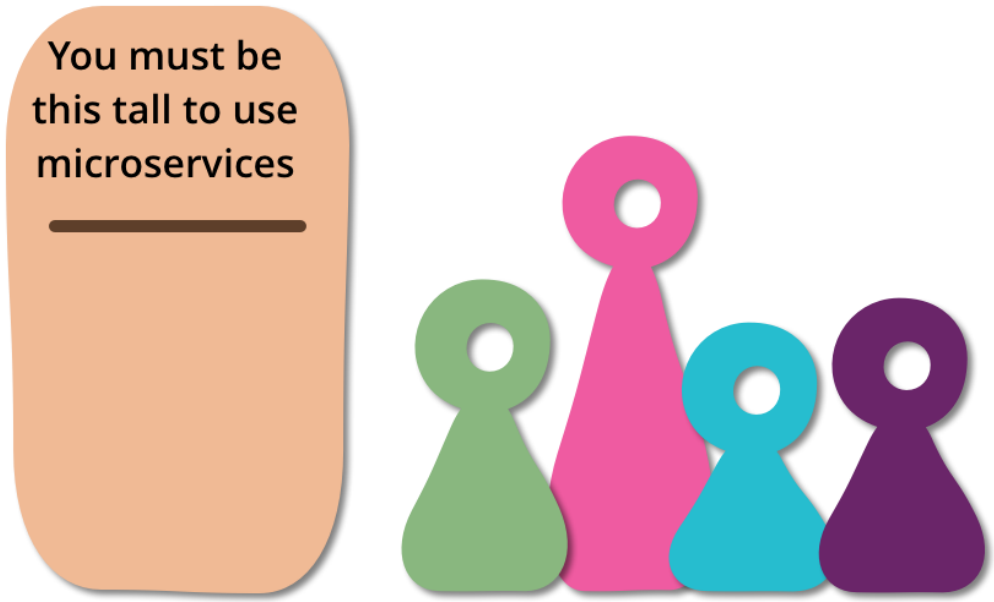
Continuous Delivery

EVOLUTIONARY DESIGN

modular design and replaceability

DESIGN FOR FAILURE

resiliency and self-healing



- Rapid provisioning
- Basic monitoring
- Rapid application deployment - DevOps Culture

https://martinfowler.com/bliki/MicroservicePrerequisites.html

ORGANIZATIONAL STRUCTURE

Any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure.

Conway's Law

CONTINUOUS DEPLOYMENT

The first 90%

- Develop
- Build
- Test
- Deploy

CONTINUOUS DEPLOYMENT

The other 90%

- Monitor
- React to problems
- Prevent problems

AUTOMATION

AUTOMATION

AUTOMATION

@DEVOPS_BORAT

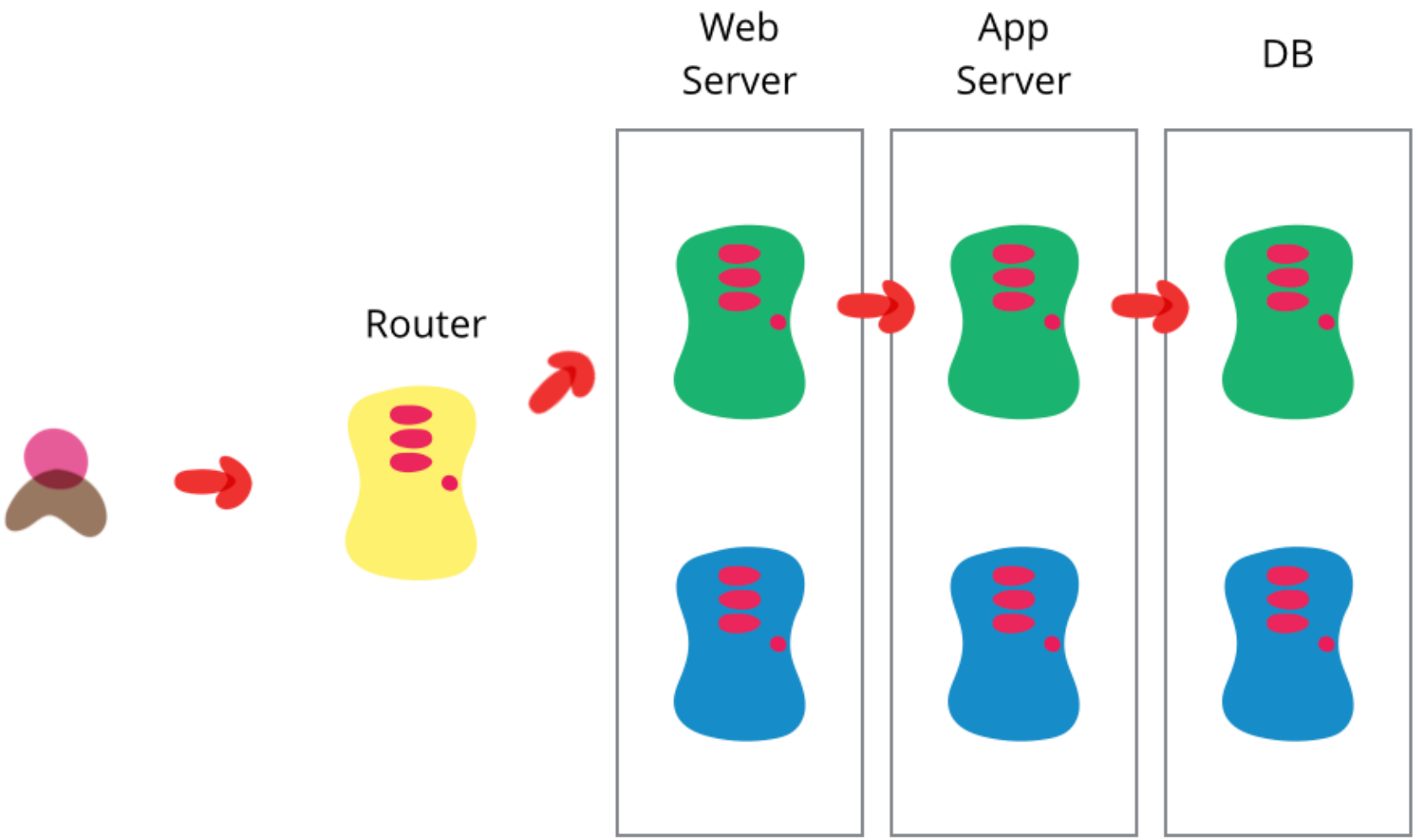
DevOps Borat

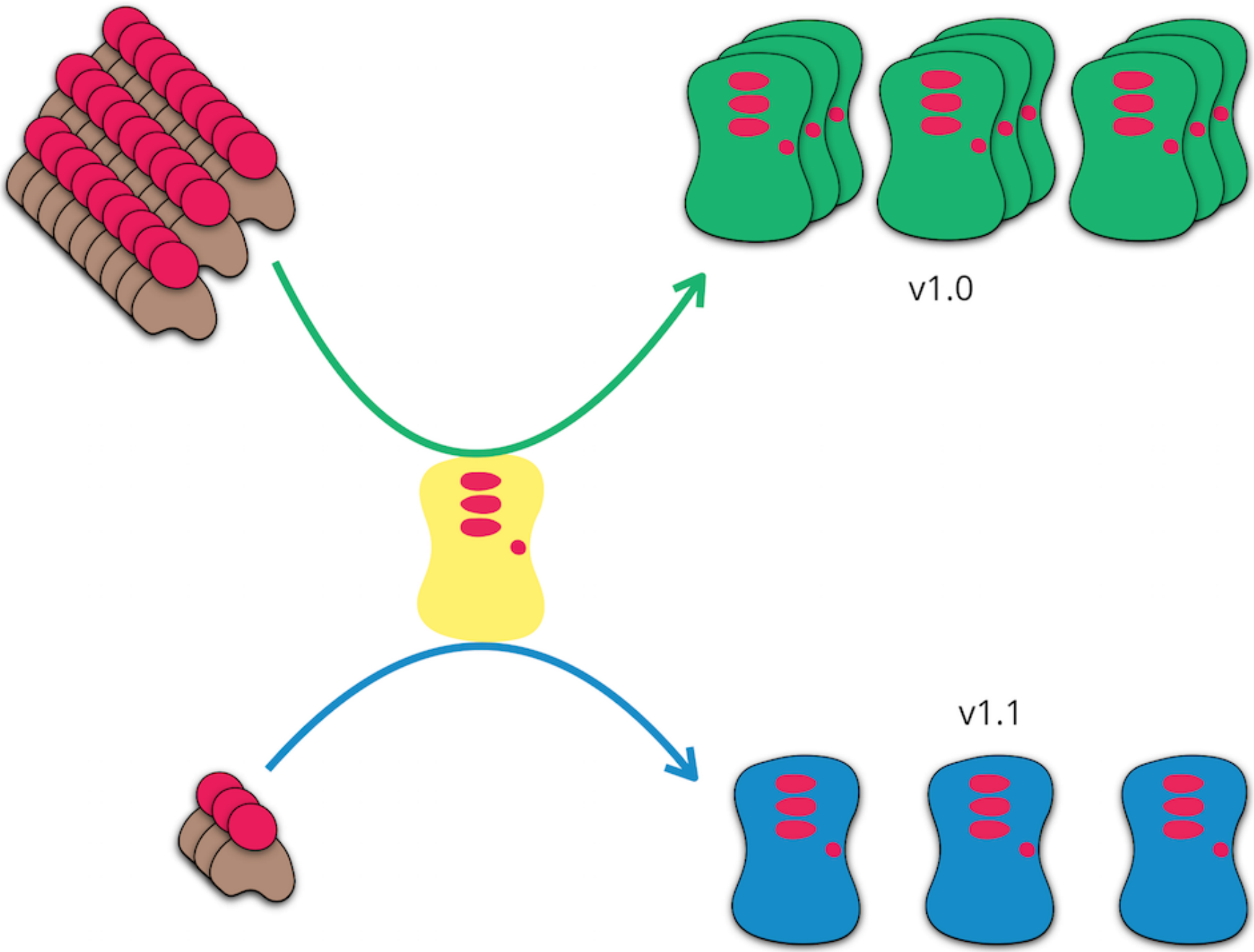
To make error is human. To propagate error to all server in automatic way is #devops.

If you haven't automatically destroyed something by mistake, you are not automating enough

DEPLOY WITHOUT DOWNTIME

- Blue-Green deployment
- Canary deployment


Web
Server
App
Server
DB
Router


v1.0
v1.1

Monitoring is the new testing

Use data from monitoring

Take proactive actions, ie. scaling

(AUTO)SCALING

New and interesting problems

EXAMPLE: AWS

Infinite capacity

EXAMPLE: AWS

Resource limits: VPCs, snapshots, some instance sizes

Rate limits: affect the whole account

EXAMPLE: AWS

Always use different accounts for testing/production and possibly different teams

Retrying is your friend, but with exponential backoff

PETS VS CATTLE

How would you design your infrastructure if you couldn't login? Ever.

Kelsey Hightower

STATEFUL SERVICES ARE HARD

Inherently

Do your services need to be deployed in a specific order?

Adding more replicas to services is not trivial

- data needs to be synced across replicas
- what if you kill a master node vs a replica

RESILIENT & SELF HEALING SYSTEMS

Services need to auto adapt to changes and errors

In case of unexpected errors, try to adapt and restore to working condition

Never expect an order of deployment

Will your app crash if database is not yet up and running?

In case your database is down, what would you do?

1. send an alert and fail fast
2. keep trying

Services need to retry calls

Can conflict with fail-fast

In complex systems there is no single cause of failure

EMBRACE FAILURE!

THE PRINCIPLES OF CHAOS ENGINEERING

principlesofchaos.org

- Build a Hypothesis around Steady State Behavior
- Vary Real-world Events
- Run Experiments in Production
- Automate Experiments to Run Continuously


amazon
web services
CHAOS
MONKEY

FIT : FAILURE INJECTION TESTING

Middle ground between isolated testing and large scale chaos exercises

http://techblog.netflix.com/2014/10/fit-failure-injection-testing.html

MICRO SERVICES AND CONTAINERS


docker

Kernel Sanders
@lstoll

The solution: Docker. The problem? You tell me.

BUT IT IS NOT TRIVIAL

CLUSTER ORCHESTRATION

Allow running services in cluster

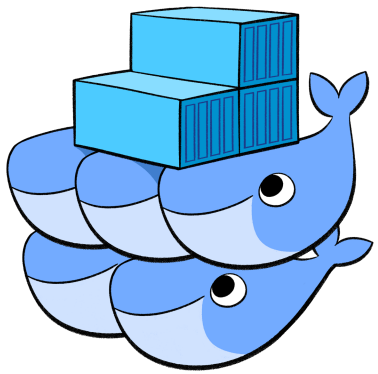
Abstract underlying infrastructure

High availability

Handle persistence for you

Network isolation and SDNs


MESOS




kubernetes

THANKS

csanchez.org

 csanchez

 carlossg

cloudbees®