Introduction To DevOps

FOR WEB DEVELOPERS

Rex Addiscentis

raddiscentis@addiscent.com

addiscent.com

github.com/addiscent

Introduction To DevOps

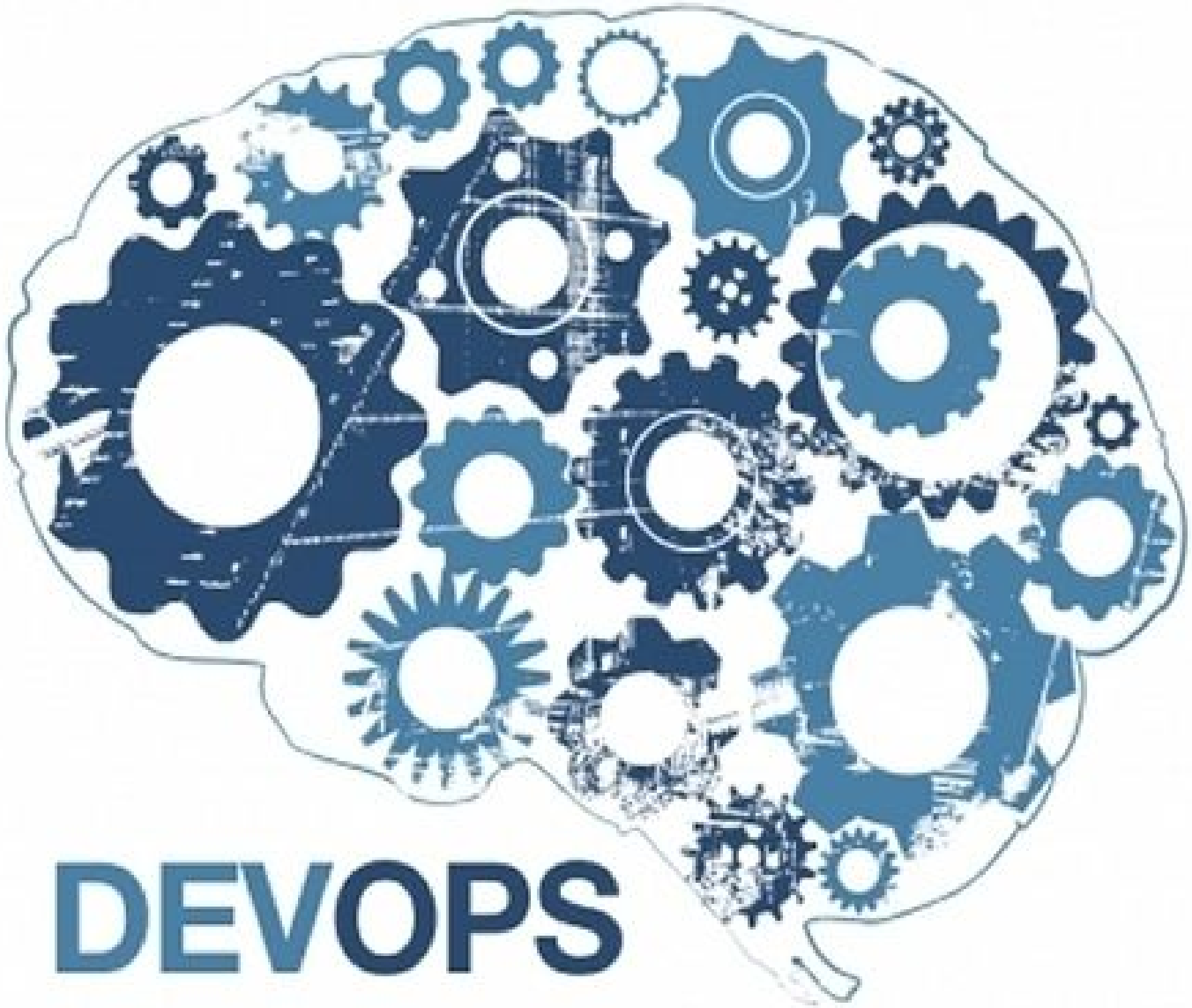
*About the **Presentation***

- “Just enough” DevOps for Demo context
- Containerization of Developers' tools not detailed
- Applicability - “it depends” on your project
    - Enterprise vs Small Business
    - Prototype vs MVP vs Mature Production
- Presentation slide deck available online, (message post on meetup web site)

Introduction To DevOps

*About the **Demo***

- Simplest-possible Dockerized PHP app
- Deployed to “Production in public cloud”
- Please save questions and note slide number for post-presentation Q&A session



Credit: cio.com

Why DevOps?

"DevOps is a *competitive advantage*"

DevOps Is A Business Solution

"Regardless of the target market, *your Company is a 'software company'*. Producing high value services software efficiently is a *fundamental part of its business model*."

Credit: forbes.com

Strategies

and

Implementations

DevOps Strategies

"Strive to **minimize** the **difference** between how and what Developers **produce**, and how and what Operations **deploys**."

Implementing DevOps

RESOURCES AND TECHNIQUES

- Cloud – IaaS, PaaS
- Virtualization - SDO
- Containers – Virtual Machines (VMs), Docker
- Microservices Architecture
- “Standardized” Developer Environments
- Continuous Deployment (with Continuous Integration)
- Automated IT (Data-center Ops)

Credit: alvareztg.com

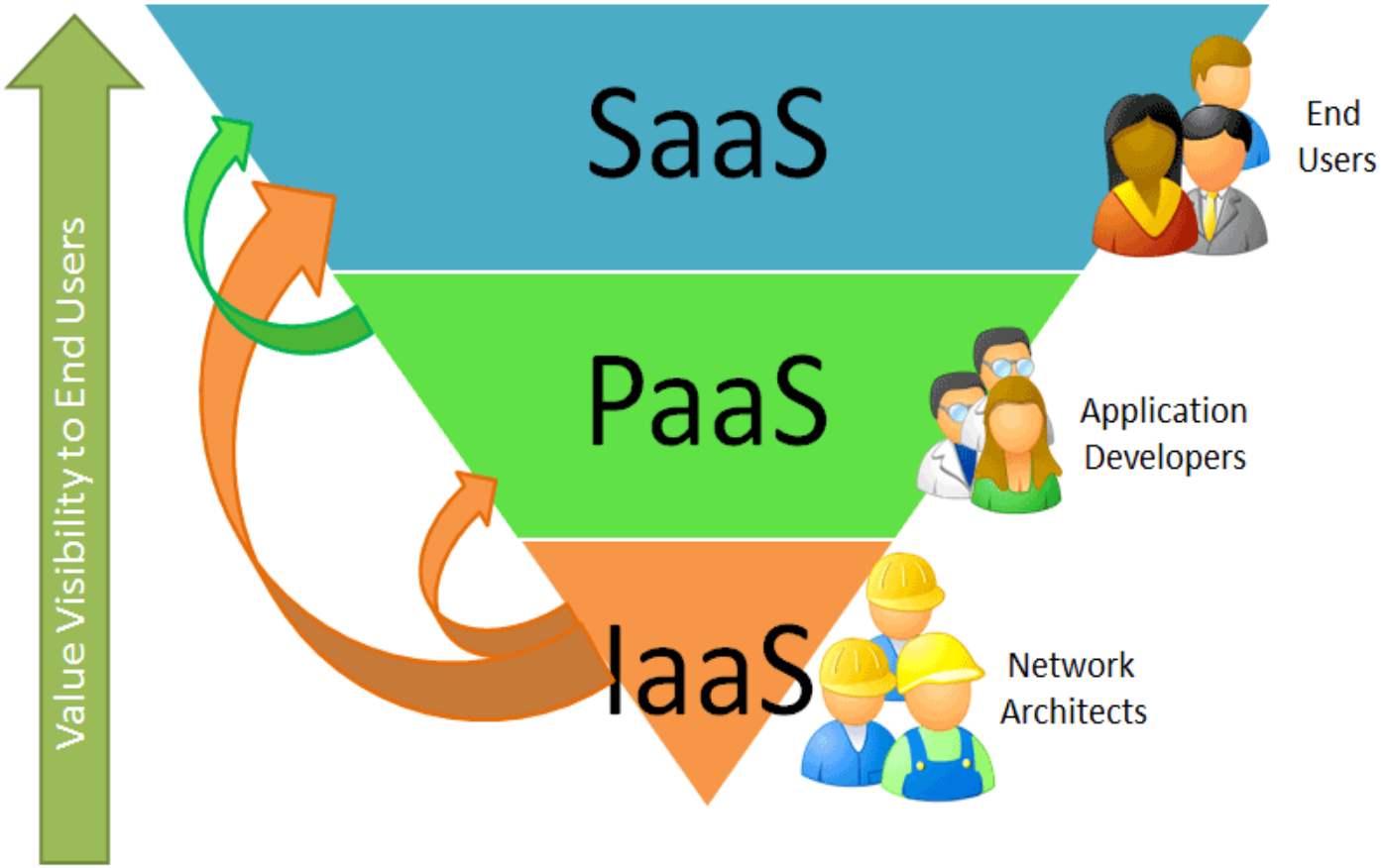
“Cloud Native” Advantages

- Cloud *Speed*
- Cloud *Scale*
- Cloud *Economics*

XaaS

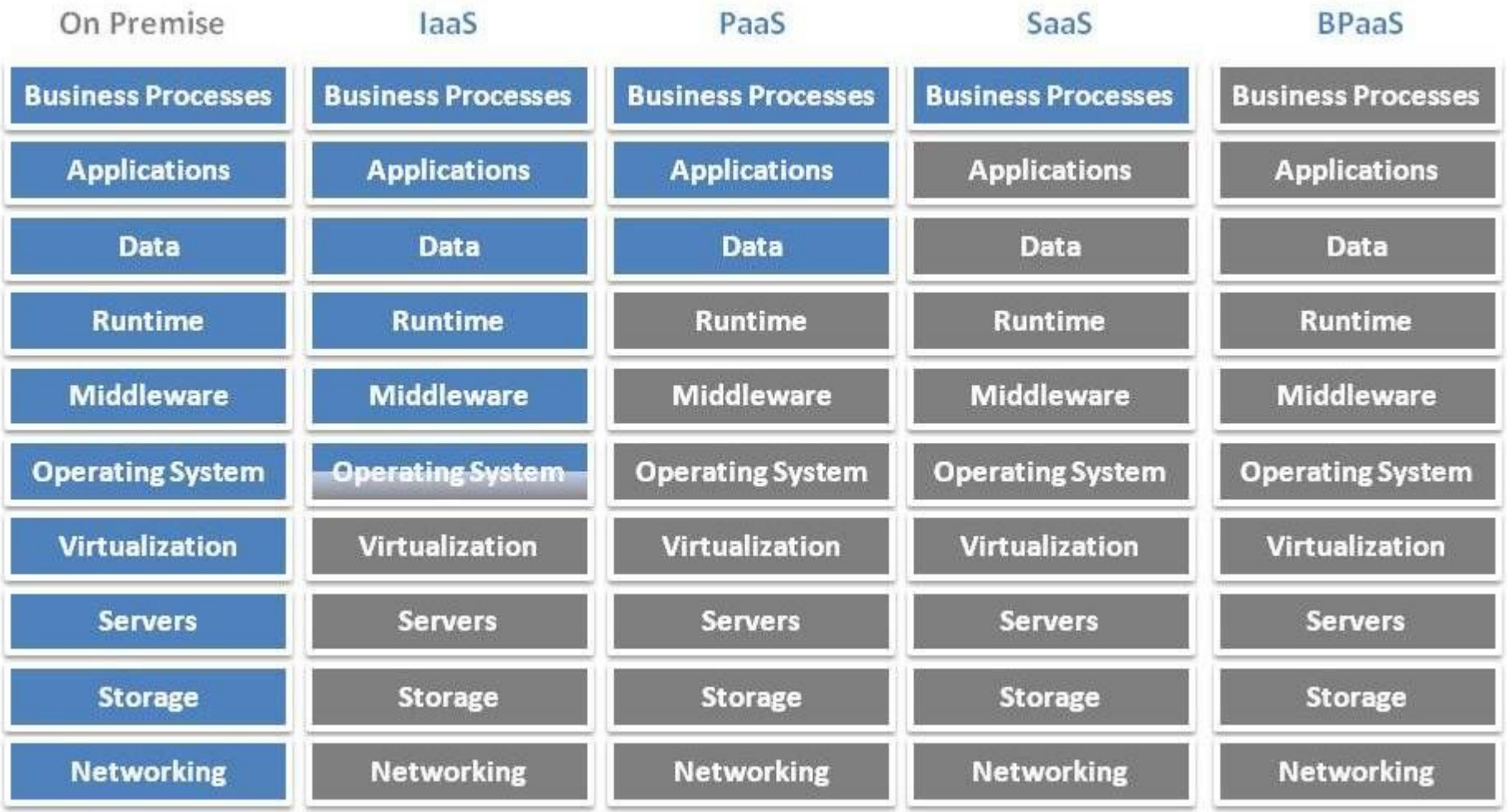
Service Categories

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (Saas)
- Anything as a Service (Xaas)
  - Monitoring as a Service (Maas)
  - Communication as a Service (CaaS)
  - Business Processes as a Service (BPaaS)
  - … etc



Credit: unknown

Cloud Service Categories



Credit: http://tecbizz.net/2012/10/everything-as-a-service-introduction/

Cloud IaaS

IaaS Computing Services

- Compute nodes
- Networking (incl VPC)
- Storage
- Containers

Cloud PaaS

PaaS Services

- Automatically handles the deployment - capacity provisioning, load balancing, auto-scaling
- Application health monitoring
- Project run-times - PHP, Java, Node.js, Python, Ruby, Go, .NET, and Docker
- Project environments - servers such as Apache, Nginx, Passenger, and IIS

Cloud Service Companies

PUBLIC

- Amazon Web Services
- Microsoft Azure
- Google Cloud Platform
- DigitalOcean
- Heroku

Cloud Services

Open Source Software (OSS)

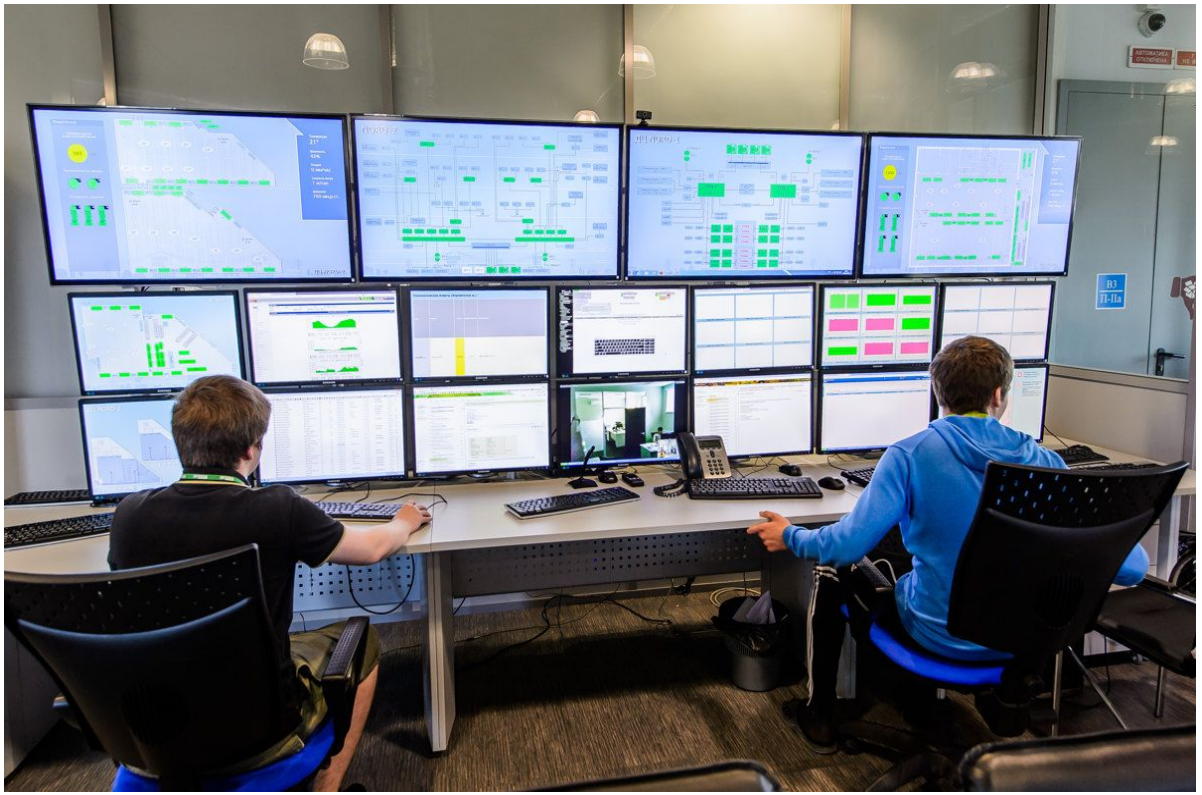
PRIVATE AND PUBLIC

- OpenStack
- OpenShift
- Cloud Foundry

Noteworthy Terms

Quality of Service (QoS): Availability-Reliability

- “NUMBER OF NINES”
- 99.999% == “Five Nines”
- High Availability == Five Nines Or Greater



Credit: http://sysmagazine.com/posts/193306/

Operations

Operations Deliverables

Examples of **Application Services Delivered**

- Online Merchant
- Social Network
- Media Provider
- Online Banking
- Publishers

Operations Deliverables

Generalization Of **Application Functions**

- Customer application-specific
  - Page Presentation (View)
  - Page Controls (Controller)
  - Function or Business Logic (Model)
- Support company's business needs
  - Marketing
  - Sales
  - Operating Expenses
- Must be "reliable enough”

Operations Activities

DELIVER AND MANAGE PRODUCT-SERVICE

Deploy

- Roll-out per constraints
- Roll-back per necessity
- Scale as needed
- Phase out end-of-life instances

Operations Activities

DELIVER AND MANAGE PRODUCT-SERVICE

Monitor

- Health
- Performance
- Cost

Operations

CATEGORIES OF FUNCTIONS AND UTILITY

Application Specific

- Inventory Views
- Accounts and Passwords
- Purchases or other transactions

Operations

CATEGORIES OF FUNCTIONS AND UTILITY

Typical Infrastructure

- HTTP service (web or app server)
- Database service
- Proxy-firewall
- TLS-SSL encrypt-decrypt
- Memcached
- Redis
- Load Balancer

Credit: stories.rackspace.com/

In a DevOps Context

Development Activities

DevOps ANALYSIS and DESIGN

- Design and Implement Software Product
- Design and Implement Development Process
- Define Operations deliverables, processes, and life-cycle management

Development Activities

DevOps ANALYSIS and DESIGN

Design and Implement Software Product

- “Ops deliverables categories” can be used as design requirements
  - Application-specific
  - Infrastructure-specific
- Define collections of "**collaborating services**", (Microservices Architecture)
- Define in detail Operations Deliverables (components)

Development Activities

DevOps ANALYSIS and DESIGN

Design and Implement Development Process

- Identify and implement “best” technologies
  - Standardized development environment
  - Continuous Delivery

Team Standardized Development Environment

- Use Reproducible Workspaces and Shared tools
  - Automate installation - containerized
  - Collaboration-friendly – establish standards and conventions, limit “free-for-all” practice when important
- Candidate tools for containerization
  - Host system, text editor or IDE, programming language interpreter (run-time) or compiler-linker, libraries/components, package or dependency managers, build/test tools, repository tools

Continuous Delivery

SOFTWARE DELIVERY PIPELINE

“Always Shippable”

- Continuous Integration
- Automated Builds
- Automated Testing, as possible
- Manual Testing, as necessary
- Marked according to Release Readiness

Continuous Integration

"Continuous integration (CI) is the practice, in software engineering, of merging all developer working copies with a shared mainline several times a day."

- Wikipedia

Continuous Delivery

Workflow and Best Practices

- Automate the build
- Maintain a code repository
- Everyone commits to the baseline “every day”
- Every commit should trigger a delivery
- Make the build self-testing
- Test in a clone of the production environment
- Make it easy to get the latest deliverables

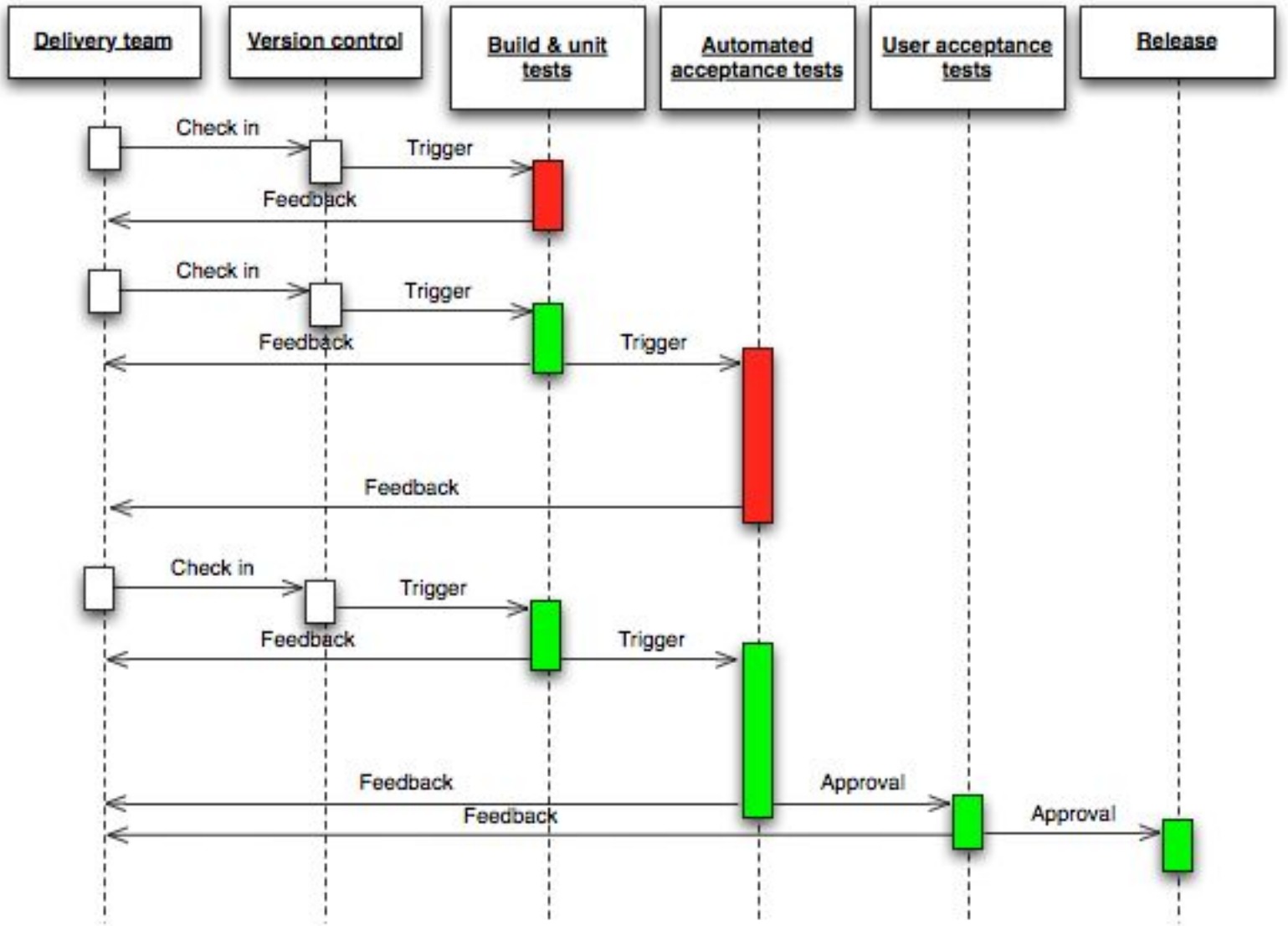
CONTINUOUS DELIVERY

Automated Build

Typical CI Coverage

- Coding Standards compliance
- Unit Tests
- ...
- Documentation generation
- Reporting

Continuous Delivery Flow



Credit: Continuous Delivery by Jez Humble and David Farley (2010)

Microservices Architecture

“OPPOSITE OF MONOLITHIC DESIGN”

Monolithic Design

- Built as atomic unit
- Deployed as atomic unit
- “Single program” can perform all possible functions
- Self-contained, independent from other platforms
- *COMPLEXITY IS HIDDEN, EXCEPT TO PROGRAMMERS*

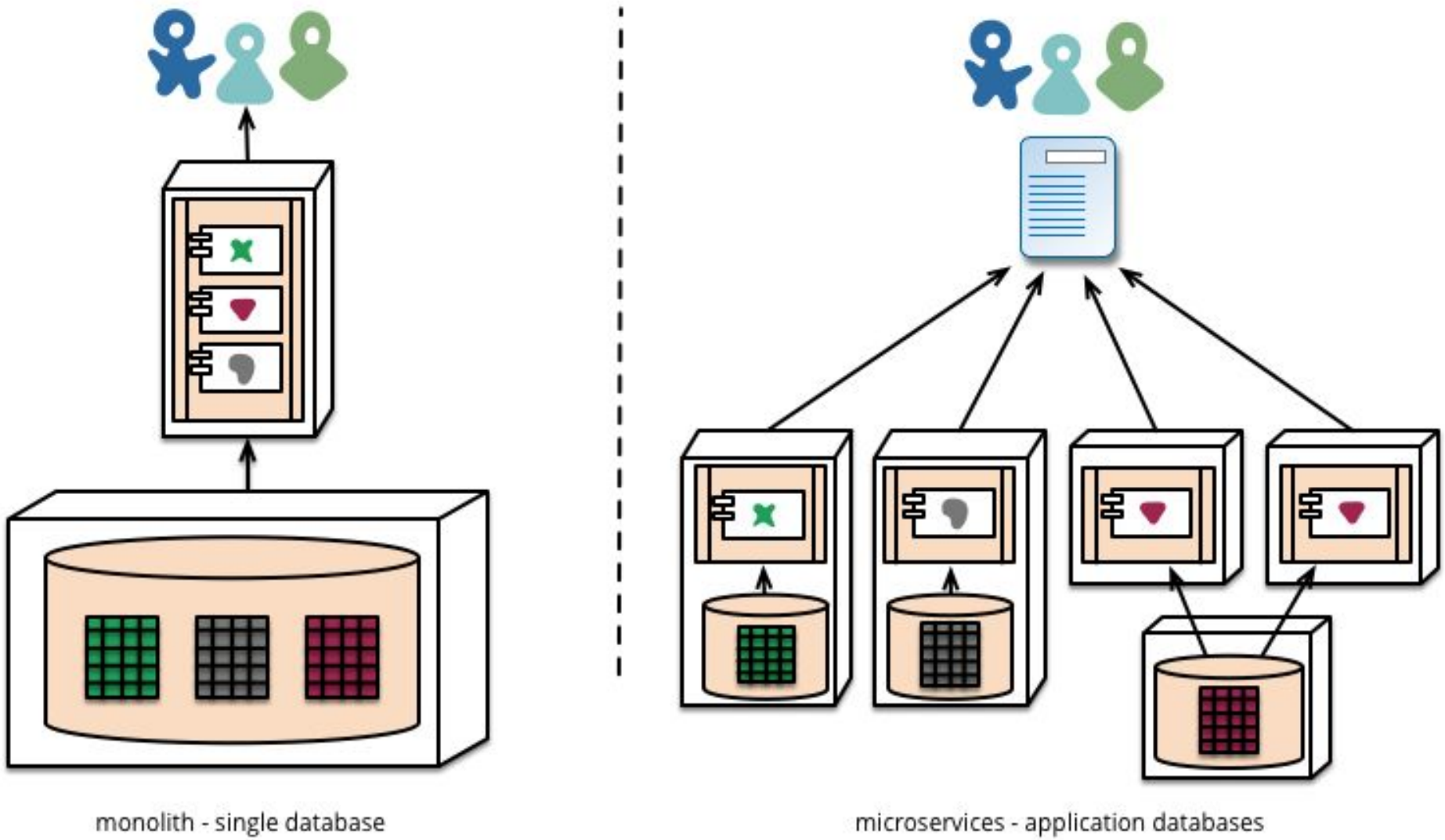
Microservices Architecture

NATURAL ATTRIBUTES

Partitioned Functionality

- Services built as “Bounded Context” units
- Services deployed as individual units
- **Collaborating entities**, division of responsibilities
- Often dependent on other services
- *COMPLEXITY IS EXPOSED*

Monolithic vs Microservices

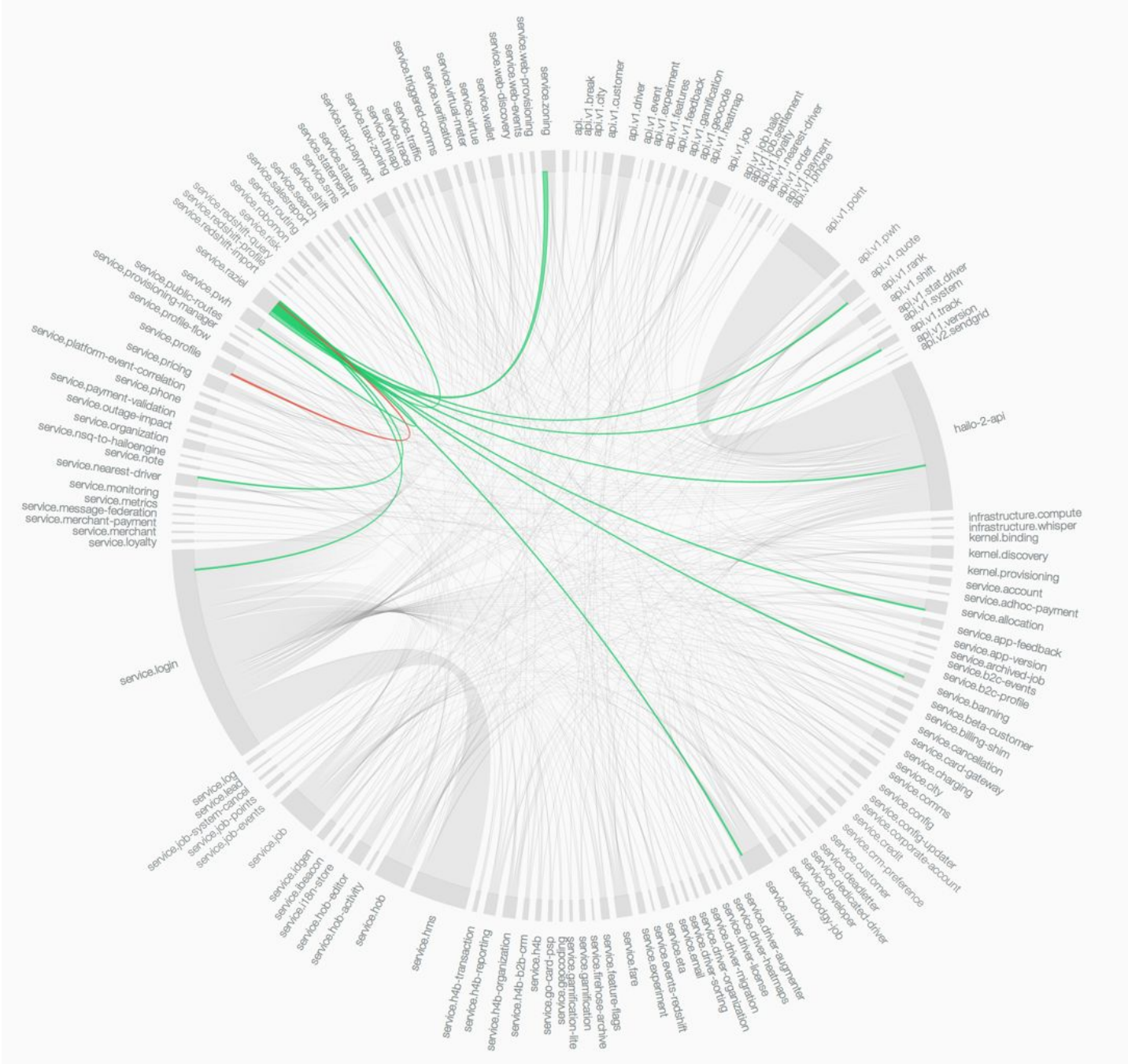


Credit: martinfowler.com/articles/microservices.html

Collaborating Microservices

Credit: thoughtworks.com

Wheel-of-Doom (Hailo)



Credit: Hailo Tech Blog - sudo.hailoapp.com

Microservices Architecture

IMPORTANT DESIGN CONSIDERATIONS

Common “Software Design Best Practices” apply to MA

- Loose coupling, high cohesion
- Object Oriented principles

Collaborating Classes

- APIs – HTTP-RESTful
- Stateless (preferred) and Stateful (when necessary)
- Bounded Domains – DDD (Bounded Context)
- “Rugged” design – Fault tolerant, (plan for system errors)

Constraints and Limits

- Minimize inter-container N-way communication (fan-out)
- Right-size – “two (four) pizza team”

Microservices Architecture

FLEXIBLE IMPLEMENTATION AND DEPLOYMENT CHOICES

Advantages

- Polyglot
- Development teams may be much smaller, (but more numerous)
- A microservice may be released on a different schedule than others
- Complexity is exposed and must be addressed explicitly, (internal "hidden" monolith complexity is revealed by split into microservices)

Microservices Architecture

NO SILVER BULLET

Disadvantages

- Complexity of Services APIs
- Complexity of Inter-Service Communication
- Complexity of Inter-Service behaviors/dependencies
- Network - Latency, unreliability, bandwidth limits
- Single large database must be split – Data consistency and integrity much more complicated

Microservices Architecture

EXAMPLES – SERVICES AS DELIVERED BY OPS

Application domain-specific

- Service presentation
- Inventory-views
- Accounts
- Purchases or other transactions
- Passwords (authorization)

Microservices Architecture

IDENTIFY THE SERVICES DELIVERED BY OPS

Infrastructure-specific

- Database service
- HTTP service (web or app server)
- TLS/SSL encryption-decryption
- Memcached, Redis
- Proxy-Firewall-Load Balancer

Conway's Law

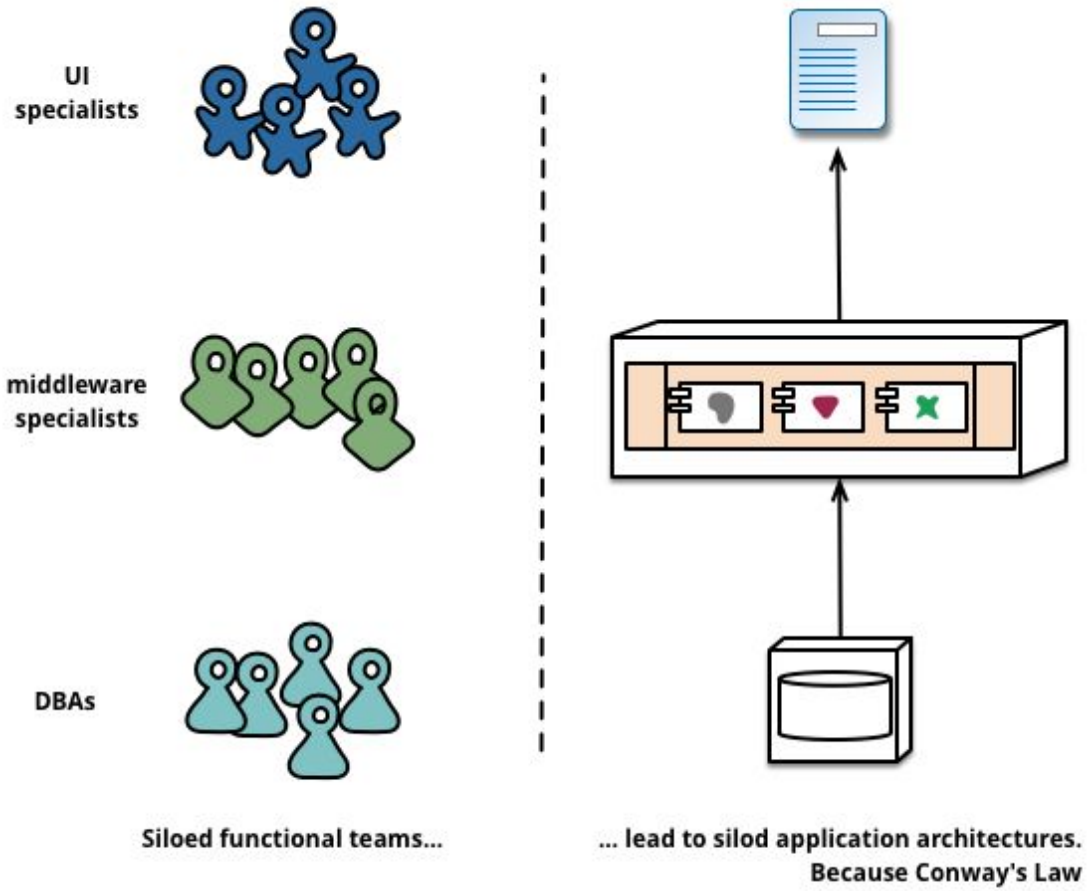
“Any organization that designs a system ... will inevitably produce a design whose structure is a copy of the organization's communication structure.”

-- Melvyn Conway, 1967

“Monolithic” Company Organization

Produces

Monolithic Architecture

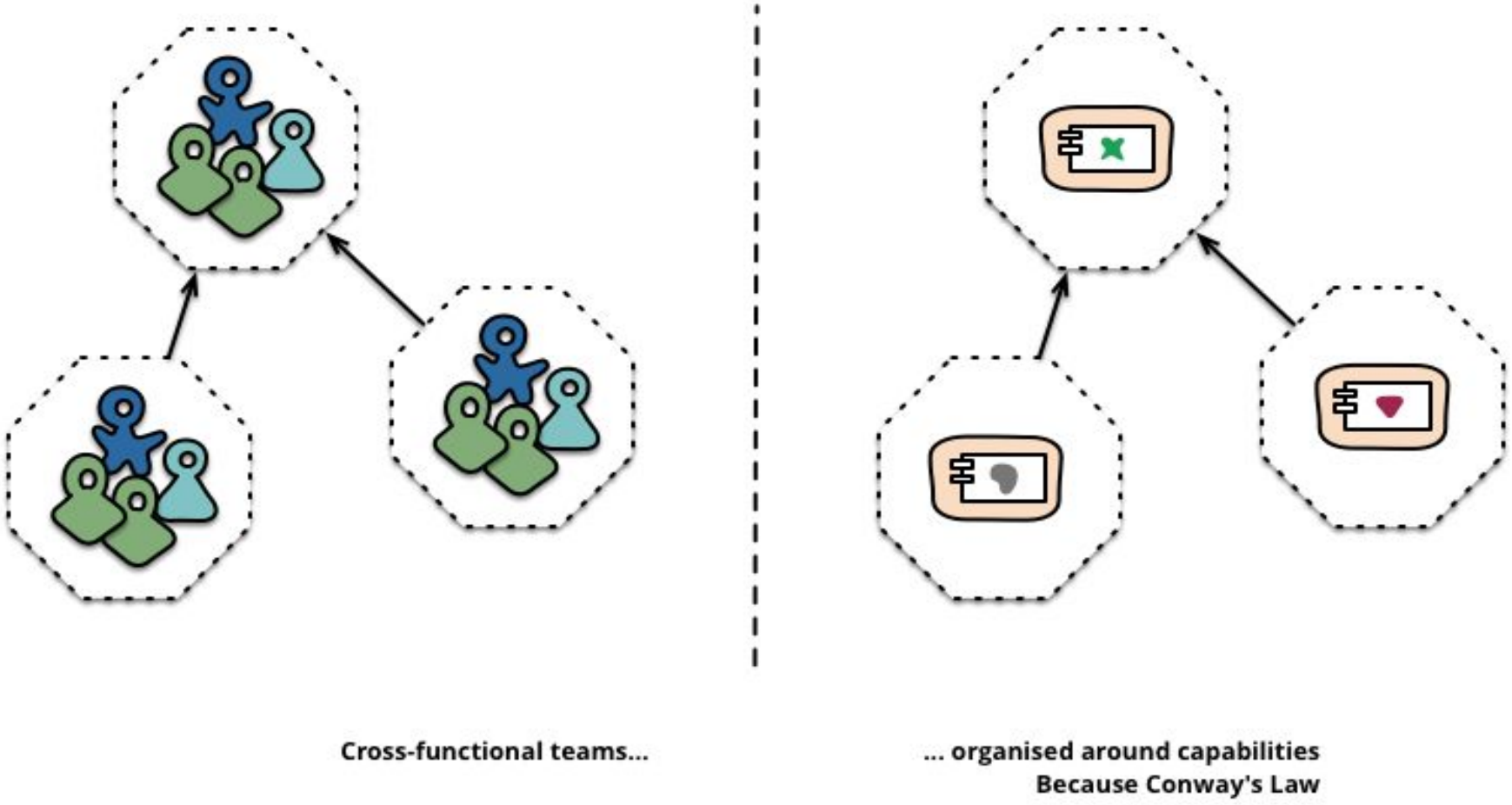


Credit: martinfowler.com/articles/microservices.html

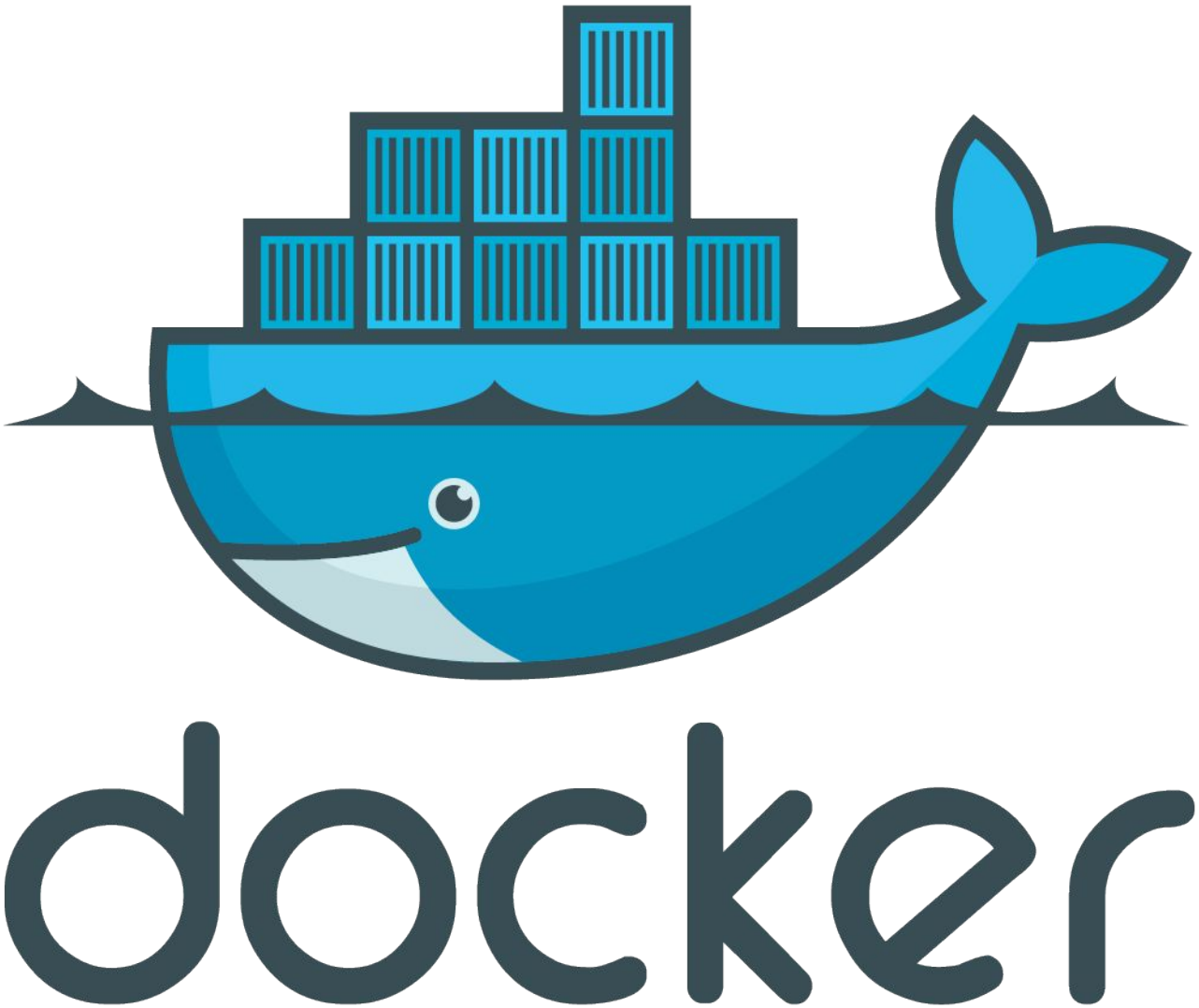
Microservices Architecture

Requires

Microservices Dev Dept Organization



Credit: martinfowler.com/articles/microservices.html



Docker

A WAY TO IMPLEMENT MICROSERVICES

"An open platform for distributed applications for developers and sysadmins."

-- docker.com

Docker

WHAT IS DOCKER?

Docker is a Company

- docker.com

Docker is a Container Technology

- Lightweight loadable binary image
- Limited capabilities – make it compelling

Docker is an Ecosystem

- Docker tools and education resources
- Third party tools and education resources

Docker

AS A CONTAINER

- Executable code
- May be data only
- Isolation of resources
- Lightweight VM-type services
- Built using Dockerfile directives

Docker

CONTAINER FEATURES

- Namespace - its own PIDs, NAT, etc
- Processes and memory are restricted
- Files system - volumes may be bound to host fs
- Network I/O - ports may be mapped
- Container Linking - inter-container communication
- Private network (VPC)

Docker

IS NOT JUST A CONTAINER

- Client command line (CLI)
- Daemon (Docker Engine) - container agent
- Repository - allows public and private
- Tools Suite

Docker

IS NOT JUST A CONTAINER

Toolset

- Docker Engine – container agent
- Docker-machine – wrangles a *host* upon which containers run
- Docker-compose – wrangles "*collaborating services*" containers
- Docker swarm - wrangles "swarm" of docker-machines, each running containers

Docker

ADVANTAGES

- Native-portable deployment across LINUX-kernel machines
- Application-centric
- Automated builds - “trusted”
- Tagging (Versioning)
- Component re-use
- Sharing
- Tool ecosystem

Docker

ADVANTAGES

Native-portable Deployment Across LINUX Machines

- Local Host – desktop, laptop
- Amazon Web Services
- Microsoft Azure Cloud
- Google Cloud Platform
- Digital Ocean
- Heroku
- Caveats for Mac and Windows hosts

Docker

ADVANTAGES

What A Docker Container Is Good At

- Image builds and loading can be fast, due to layers and caching on local host
- Small memory and storage requirements
- No ***processes*** overhead specific to container

Docker

MIS-USAGE

- Base image which is unnecessarily “fat”
- Too many processes in one container
  - **Anti-pattern: Dockerized LAMP stack is NOT a wise micro-services pattern for production**

Container-related Strategies and Techniques

"BEST PRACTICES" SUBJECTS FOR FURTHER STUDY

- Immutable containers – Don't modify instances
  - Don't ssh into production VMs or containers and modify their state or configuration
- Ephemeral – "Design for Transience"
  - Accomodate short lifespans
- Tag (version) containers, build using tags
- Use Load Balancers for rollout, rollback, canary cages, and scaling
- Starting as Monolithic vs starting as Microservices - "It depends"

Amazon Web Services

aws.amazon.com

- AWS Elastic Compute Cloud (EC2)
- Compute nodes created/destroyed as needed
- Compute nodes may be long-running, or short-lived
- Spin-up time scale = ~minute(s)
- An EC2 node has no storage – must use S3 or other AWS DB services, or other networked product