Shopify

— **Commerce platform**: Online stores, POS, Facebook, ..

— Can handle at least 10,000 RPS, 10,000 Orders per Minute

— 300 million unique monthly visitors

— 5.3 million orders per month

— 4000+ containers on metal in two data centers

Make it the best place in the world to **solve commerce**

IaaS to PaaS to CaaS

Docker in production for **main application** and some services for **one year**

Adoption triad

1. CI/Development
2. Production
3. CaaS

Shopify's Timeline

We did it in reverse. Not recommended.

— **2014 Jan-Jun** Attempts of CaaS with CoreOS/Mesos. Initial brickwork.

— **2014 Jul** Docker powering 100% of Shopify web with boring tech

— **2014 Aug-Dec** Fire fighting, preparing for holidays

— **2015 Jan-May** Deploys to 3m, CI to 5m

Greenfield *vs* Legacy

Infrastructure doesn't have an end state. It's constantly evolving.

Vendors are focused on **Greenfield**

Shopify and many companies focused on **Legacy**

The only agreed upon interface is Docker.

Development/CI is where pros currently outweigh cons for some cases

Invest time? 1-5	Preperation	Dev	CI	Production	CaaS
Monolith	3	2	2	1	1
Services	5	3	4	2	2
Security				1	

Roadmap to Production to CaaS

What I wish someone had told us.

1. Exploration
2. Preparation
3. Containerization
4. Feature/CaaS

1. Exploration

What is it and do we need it today?

— Study `namespaces(7)`, companies, `cgroup(7)`, ..

— Develop feel for containers

— Don't get stuck or starstruck

— Rapid prototyping--throw everything away after

2. Preparation

— You can't just shove an existing
application inside containers.

— Evolve your current stack.

2. Preparation: Overview

Mindset of immutable. Think of your application as **a binary**.[1]

— Secrets

— Logs

— IPC via network

— Deploys, scale by adding containers, .. environment specific what needs change

[1] http://12factor.net/

2. Preparation: Explicit non goals

— Containers

— Change routing

— Switch Linux distribution

— Orchestration

2. Preparation: Secrets example

— Configuration management common

— Secrets should either live in the image or be requested externally

— Shopify built `ejson` (lives in image)

— Hashicorp recently released `Vault` (requested externally)

2. Preparation: Logging example

— Don't attempt to log to files inside containers (what we did)

— Vendors haven't caught up yet (e.g. Splunk)

— Logging drivers landed in 1.6.0

— Many different solutions around: syslog container, network, stdout/stderr, ..

3. Containerization: Overview

Focus on retaining current feature set, not adding new.

— **Get really confident with containers.**

— Development/CI

— Everything but containers should stay the same

— Production?

3. Containerization: Getting good at containers

— Building container images

— Union filesystems

— `init` in container

— Monitoring

— Edgy kernels, security updates, registry, ..

3. Containerization: Explicit non goals

— Change orchestration

— Change deployment

— Change Linux distribution

— Change anything but containers, or go back to 2

3. Containerization: `init` example

— Zombie processes in containers don't get acknowledged by default

— Can lead to interesting kernel scenarios when too many zombies accumulate

3. Containerization: Building images example

— Shopify was not able to get `Dockerfile` builds fast enough

— Built our own image infrastructure

— Docker needs to expose primitives

3. Containerization: Union file systems

— Docker needs a file system that allows for CoW

— AUFS, BTRFS, ZFS, Overlay, ..

— We've tried them all in production, and until Overlay it was extremely painful

4. Feature/CaaS 😍

Don't let anyone sell you this today.

— Distributed orchestration

— Buttons

— Completely consistent deployment stack

— Minimal Linux distributions

— Docker as `init`

Containers are the future. But it's still hard.

CHICAGO

INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2015

goto;
conference

Questions?
Please remember to evaluate via the GOTO Guide App