Evolution of an exchange

22. July 2023

Alen Krmelj
Cloud Infrastructure Team Lead

Bitstamp

Presentation Purpose

01

What is the purpose of this presentation?

Journey is as important as the end result. Few things we had in mind during all these years improving our infrastructure:

Divide and conquer – Approach a big problem and break it down into smaller pieces

Move fast in small chunks – small wins every short while, are better than years of development and no progress on current state

Critical thinking – Not every technology that others are using is appropriate for your use case

Vision – Always think ahead, what's the next step; What are the pain points of current state and what would be short and long-term solutions

Initial State

02

Initial state

State of the infrastructure:

- Infrastructure copied 1:1 from on-premise servers
- Monolithic application
- Bunch of bash wrappers and scripts
- Deploy procedure is basically SSH to servers and reload stuff
- Beta servers on production for testing

AWS services used:

- EC2 (Various OS-es)
- ElastiCache (Redis & Memcache)
- RDS – MySQL (+Binlog read replicas)
- NFS (Mounted on EC2 instances)

Problems:

- Deploy procedure
- Developers need stagings
- Configuration mess
- Vague server roles

Triage Phase

03

First steps…

Changes:

- **Forming of first SysOps department**
- **IaC** – Introduction of Deploy Toolkit (DTK) - Bash/Python framework
- **Definition of roles** – Started working on defining what servers are running
- **CloudFormation** – Troposphere Python lib to generate it
- **CodeDeploy** – Server installation code in Bash
- **Golden AMI builder** – Our own implementation of EC2 AMI building tool, part of DTK

Gains:

- Stagings for developers
- Deploy procedure standardized
- Server roles – configuration mess solved

Problems:

- **EC2 instance provisioning**
  - Bash is terrible
  - Developers want to deploy stagings on their own
- **CodeDeploy**
  - Slow
  - Bunch of EC2 Tags (All, General, Role based tags, etc…)
  - Our complexity of deploy procedure doesn't fit well
  - Blue/Green deployments breaking our CloudFormation

Improvement Phase

04

Slowly getting there…

Changes:

- **Ansible** – Replace all Bash with ansible
- **Automation Controller** – Formerly known as Ansible Tower to give devs ability to deploy
- **AWS EC2 Image Builder** – Ditched our own golden AMI builder
- **MySQL -> Aurora** – Reduced lag of readers tremendously
- **AWS SSM** – For improved passing of infra settings to application (Deploy tags etc.)
- **AWS AutoScaling** – Now possible, cause we have state in SSM
- **AWS SCM** + **SCS** – No more secrets on instances, cache secrets and prevent access to them
- **Remove NFS** – Shared storage should be S3

Gains:

- Devs can now deploy on their own
- Deploy procedure now fits exactly as we want it – flexible, parallel and fast

Problems:

- Devs are reluctant to learn DTK
- Testing of DTK full blown stack takes time
- Microservices – Company decided microservices approach, meaning current setup would become tedious to maintain
- Devs are using containers – we need to adapt

Transformation Phase

05

Empowerment is the key…

Changes:

- **AWS Organisations** – Redesign account structure, get the ability to give accounts to devs, have control over billing and limit blast radius
- **AWS RAM** – Redesign AWS networking structure, get ability to share resources between accounts
- **AWS ControlTower** – Enforce same policies everywhere
- **AWS Nework Firewall** – Create isolated stagings
- **AWS SSO** – Connect it to our AD and move away from IAM users
- **Terraform** – Move away from limitations of CloudFormation, speed up infrastructure deploy and testing along with policies
- **Bitstamp Application Framework (BAF)** – A successor to DTK, using Python version of CDK-TF
- **AWS ECS** – Slowly moving away from EC2 instances where possible, use containers prepared by devs
- **AWS MSK** – Kafka as glue between microservices

Gains:

- Devs become owners of their services and get confident with deploys
- Full blown CI/CD implementation can be made
- Simplification of code

Optimisation Phase

06

More performance, more resiliency…

Planned changes:

- **AWS Cloud Map** – Introduce AZ aware applications, ensure minimal latency
- **AWS EKS** – As we see it only cost reduction technology, we might use it for some specific use cases, but as of now ECS is much better choice
- **AWS CloudFront + AWS WAFv2 + AWS Shield** – We already use these on specific endpoints, but depending on the deal with AWS for traffic costs, we could switch to it as our main CDN instead of Incapsula Imperva. Main reasoning would be edge websockets scaling
- **AWS Outpost** – Depending on the decisions of upper management we might consider also moving part of our infrastructure into AWS Outpost
- **AWS MemoryDB** – Needs to be reviewed if certain edge cases are better handled in MemoryDB instead of Redis

Replaced AWS services

07

All that glitters is not gold… even on AWS…

Many times something that seems like a great solution, in reality, it just doesn't fit well.

CloudFormation

- Hard limit of 500 resources per nested stack
- Passing arguments via nested stacks is terrible, no support for objects
- No negative testing like terraform compliance
- Internal server errors at execution
- Vague errors – CF says something, real reason somewhere else
- Dependency between services problems on AWS side (like NFW not destroying properly etc.)
- Change sets are kinda ok, but Terraform Plan is better

CodeDeploy

- Agent installed on servers
- Tags here, tags there, tags everywhere – total mess to create several groups of deploys
- Blue/Green deploy breaks CloudFormation
- Slow to execute things
- Poor control over how things are actually executed

ElastiCache Memcache

- Don't use it. Please.
- Unless you know what you're doing
- Client version upgrades, loss of data upon restart, data hashing based on nodes, etc…

Thank You