SERVERLESS
&
PAAS

Wojciech Barczyński
wojciech.barczynski@wsb.wroclaw.pl

Wykład & ćwiczenia

- Materiały: github

Środowisko

Rekomendacja:

- Ubuntu 20.04 albo inna dystrybucja / *nix - MacOS
- Dual-boot?
- Lang: Python, Golang, dotnet core, ...

Ćwiczenia

- Problem
- Mierzymy się z zadaniem samemu - timebox
- Pytamy / prosimy o pomoc.
- Zrobiłam / Zrobiłem - warto pokazać
- Bez Kopiuj&Wklej (chyba, że wykładowca powie inaczej)

Jak wybrać?

Jeszcze jeden aspekt

Ostatnio było o metrykach,
dzisiaj o:
Złożoność setupu

Następnym razem

O najważniejszym (**why**): biznes

Ważne

- Zacznij od "najpierw Tak",
  wysłuchać, wspomóc w wyrażeniu pomysłu
- Risk / Reward
- Spisać argumenty za i przeciw

Ważne

- Make it real.
  Ideas are cheap (heroku)
- Strategie: v1/v2 lub PoC, ...

Najpierw TAK

- Improwizacja
- Poprawia kulturę w firmie

Złożoność setupu

- Szybkość startu nowej osoby (time to first commit)
- Efektywność Onboarding procesu

Złożoność setupu

- Ilość technologii z którą dew musi pracować

Ilość technologii

Możliwe rozwiązania:

- monitorować czy sytuacja się nie pogarsza
- kategoria oceny przy wyborze rozwiązania
- zespół developer experience / platform

Złożoność setupu

- Warstwy abstrakcji
- "Magiczne" narzędzia
- Zapomniane warstwy

Złożoność setupu

- Lokalne środowisko deweloperskie

Złożoność setupu

- Onboarding process
  oraz Engineering Guide
- Brak wspólnych konwencji

Ostatnie zajęcia

Logowanie do VM

1. Zawsze ssh z kluczami
2. Przez bastion
3. AWS SSM / AWS EC2 Instance Connect

Serverless

Korzyści

1. Brak maszyn
2. Pay-as-you-go

Ograniczenia

1. Cold-start (np., dla AWSa)
2. Ograniczenia czasowe i zasobów (limits)
3. Trudniej debugować w produkcji

Technologie

1. AWS Lambda
2. Azure Functions i Google Cloud Functions
3. CloudNative: KNative
4. @Edge, np., Cloudflare czy AWS Lambda@Edge

AWS


Connected Users and Streaming Dashboards
Web and Mobile Applications
IoT Devices
Private Applications: VPC and On-Premises
Amazon API Gateway
Create, publish, maintain, monitor, and secure APIs at any scale
API Gateway cache
Amazon CloudWatch
AWS Lambda
Amazon EC2
Amazon Kinesis
Amazon DynamoDB
Other AWS services
Publicly accessible endpoints
Private Applications
VPC and On-Premises
Data Center

AWS

Wybrane z AWS:

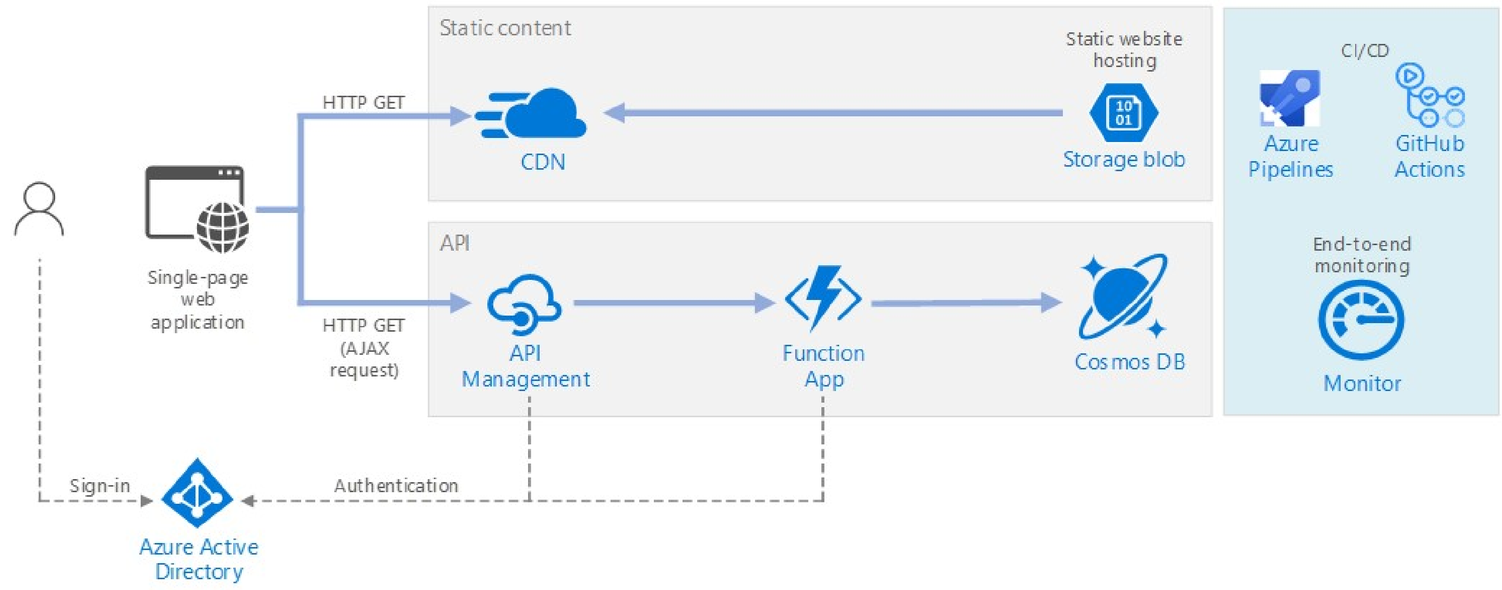
1. AWS Lambda
2. AWS Fargate
3. AWS API Gateway
4. AWS SNS, SQS, Dynamodb, EventBus
5. AWS Step Functions
6. Logs (AWS CloudWatch) and tracing (AWS X-Ray)

AWS

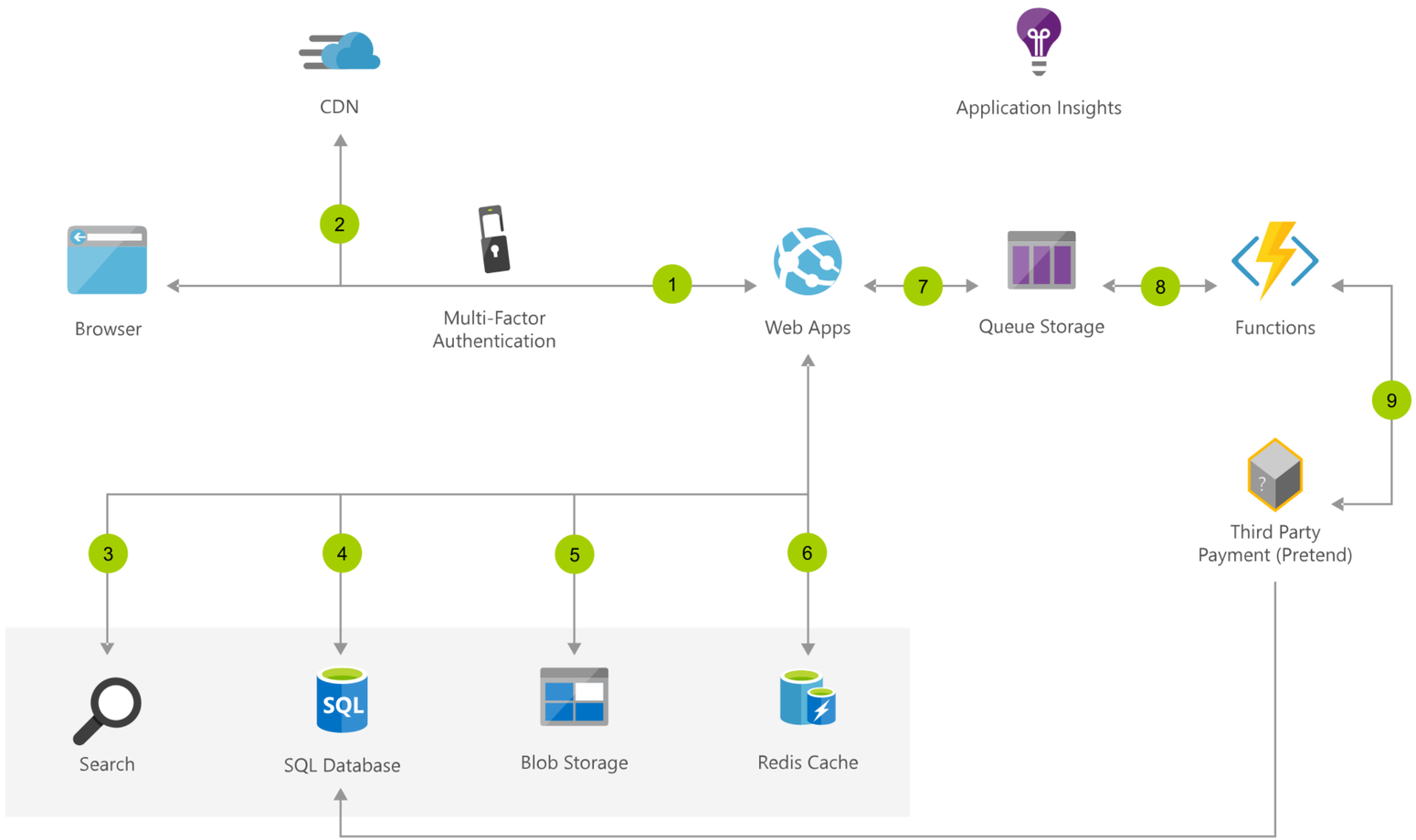
Narzędzia:

1. AWS SAM
2. Serverless
3. AWS Cloudformations, AWS CDK lub Terraform dla zasobów wspierających
4. localstack

Azure


Static content
Static website hosting
HTTP GET
CDN
Storage blob
CI/CD
Azure Pipelines
GitHub Actions
API
End-to-end monitoring
Single-page web application
HTTP GET (AJAX request)
API Management
Function App
Cosmos DB
Monitor
Sign-in
Authentication
Azure Active Directory

Azure


CDN
Application Insights
2
Browser
Multi-Factor Authentication
1
Web Apps
7
Queue Storage
8
Functions
9
Third Party Payment (Pretend)
3
4
5
6
Search
SQL Database
Blob Storage
Redis Cache

Azure

Wybrane z Azure:

- Azure Functions
- Azure API Management
- Azure EventGrid
- Azure SQL Datavase (serverless)
- Observability: Azure Monitor

Azure

Wybrane z Azure:

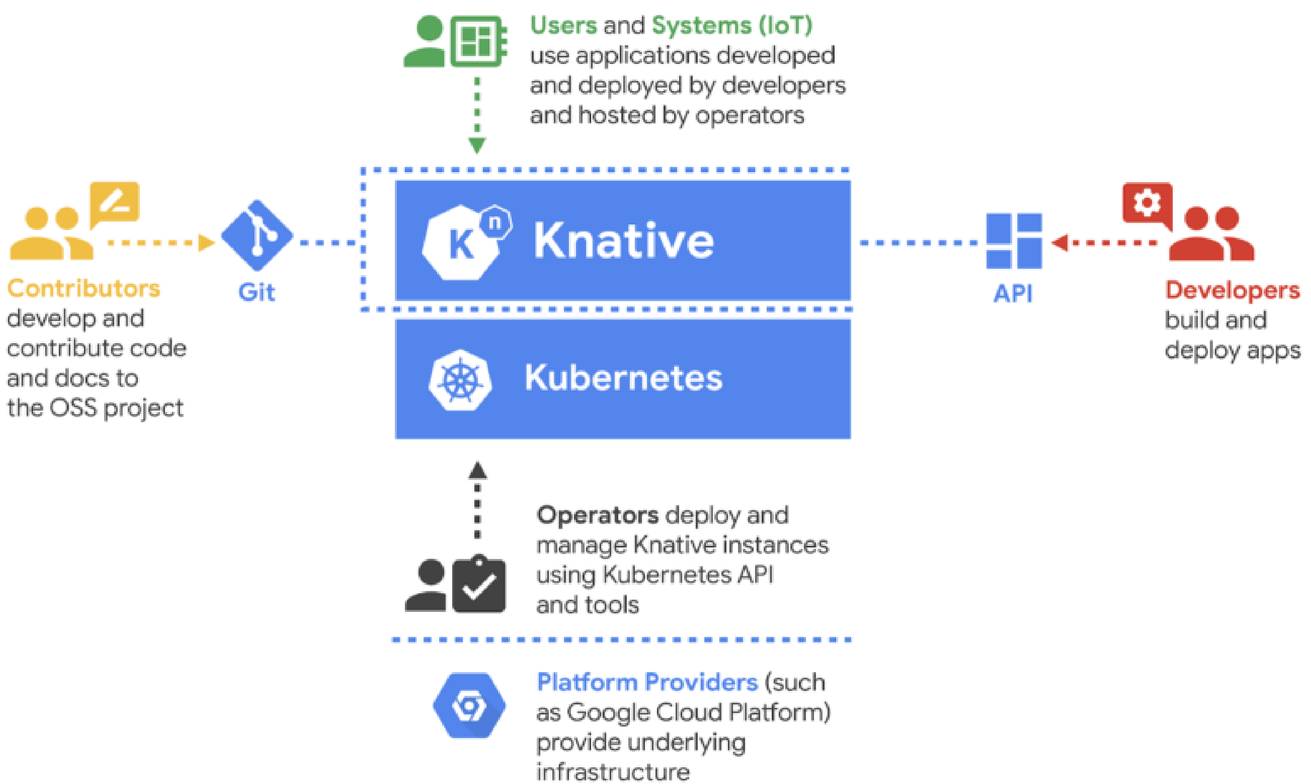
- Azure Functions
- Azure API Management
- Azure EventGrid
- Azure SQL Datavase (serverless)
- Observability: Azure Monitor

Azure

Narzędzia:

- Azure Functions Core Tools i Azure CLI
- serverless
- terraform/terragrunt dla zasobów wspierających

CloudNative - knative

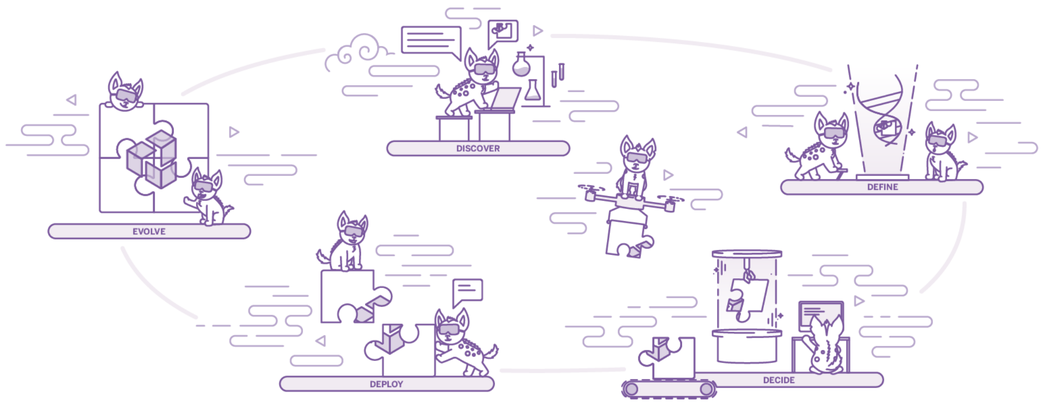


źródło

CloudNative - knative

- onPremise lub self-hosted (oferowane przez GCP)
- pozwala dynamicznie zarządzać infrastrukturą dla zmiennych workloadów

PaaS



źródło

Korzyści

- Szybkość
- Prostota
- Nie trzeba myśleć o infrastrukturze (do czasu)

Wady

- koszt
- ograniczenia
- "za dużo magi" szczególnie przy większej skali

Nie tak popularne

Wypierane z jeden strony
przez CaaS (Container-as-a-Service)
lub wewnętrzne implementacje platformy,
a z drugiej przez serverless.

Scenariusze

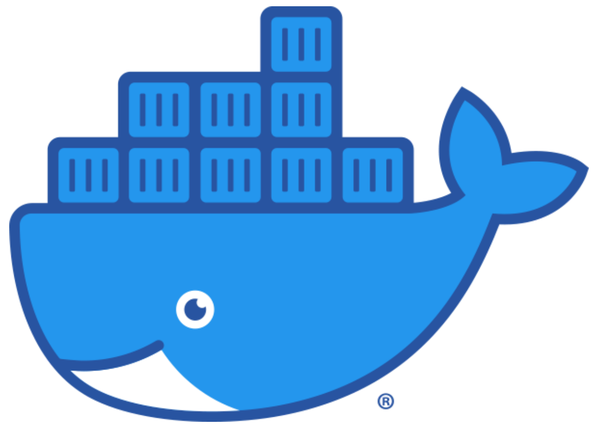
1. MVP
2. W mniejszej skali
3. Platforma pluginów dla SaaS
4. Aplikacje skupione na web frontendzie, patrz sukces
   netlify

Platformy

1. Heroku
2. Google App Engine
3. Netlify
4. Azure App Service
5. Część oferty AWS, można traktować jak PaaS

Kubernetes

Docker



Docker

Problemy:

- "u mnie działa"
- dependency hell
- jak to uruchomić :/?

Dependences

- A -> B -> C 1.0 and A -> C 2.0 💀
- biblioteki applikacji
- biblioteki natywne

Pakiety

- deployment z githuba
- pip/poetry
- deb/rpm
- fat packages (apt, runtime, ...), e.g., dh-virtualenv

Pakiet: wirtualne maszyny

- Plusy: ultimate fat packages
  wszystko od bibliotek applikacji do natywnych
- Minusy: duży rozmiar, wolno się budują

Immutable infrastructure

Pakiet: docker

- coś jak VM ale szybkie w tworzeniu i deploymentu
- filozofia Dockera - ma działać w domyślnych ustawieniach
- lekkość procesu

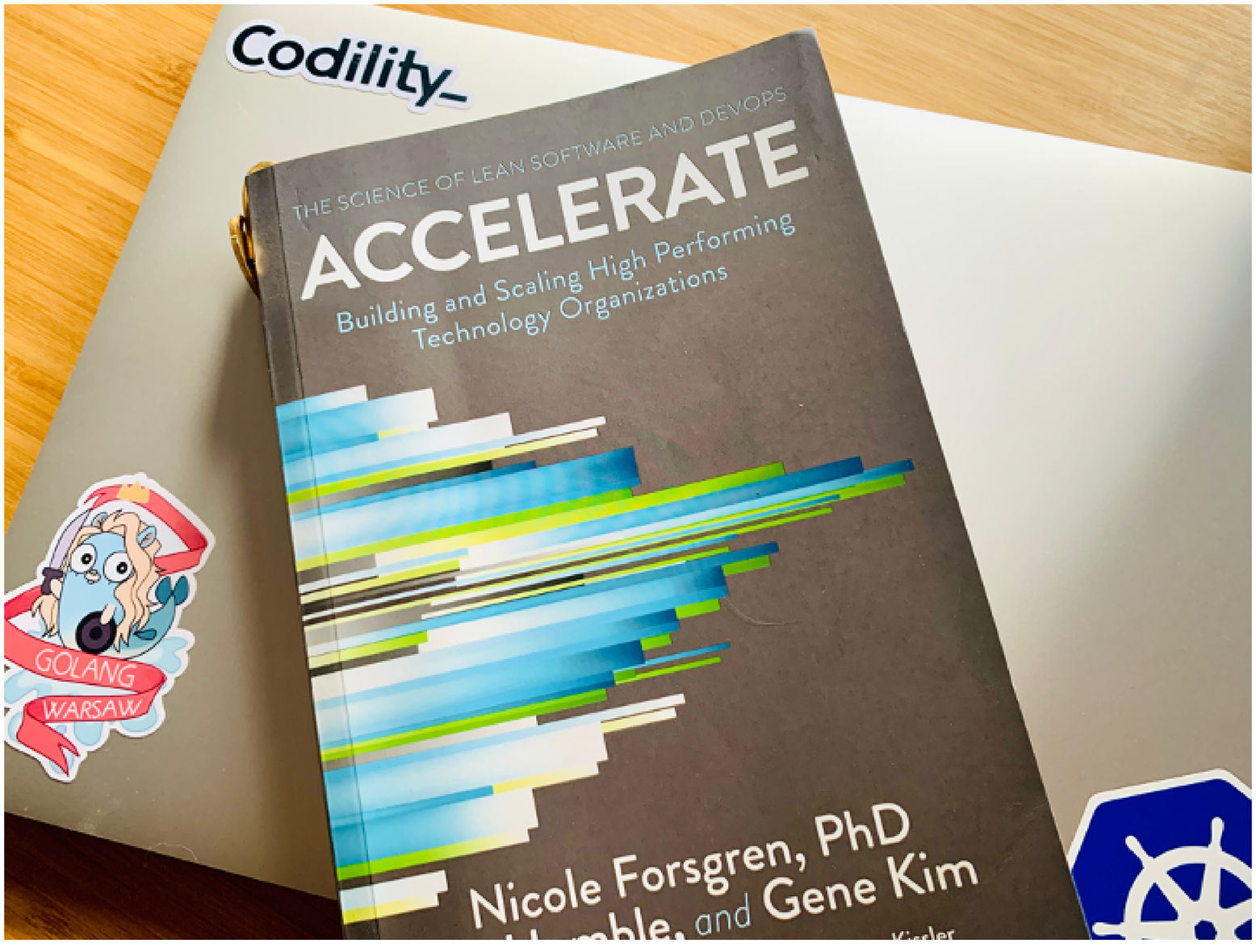
Kubernetes

- traktowanie naszego klastra jak czarną skrzynkę
- jeden język dla opisywania jak nasza aplikacja jest uruchomiona
- batteries included

Kubernetes

- Kubernetes workshop

Dziękuje za uwagę



Backup slides

Narzędzia

Misc:

- statuscake / pingdom
- locust
- zimwiki