#TrendingTopics

Serverless e SRE
TIJGS - 1o Meetup Dev Day

Agenda

- ❑ $whoami
- ❑ Evolução da computação
- ❑ Arquitetura Serverless
- ❑ Site Reliability Engineering
- ❑ Dúvidas

$whoami

/matheusrossato/

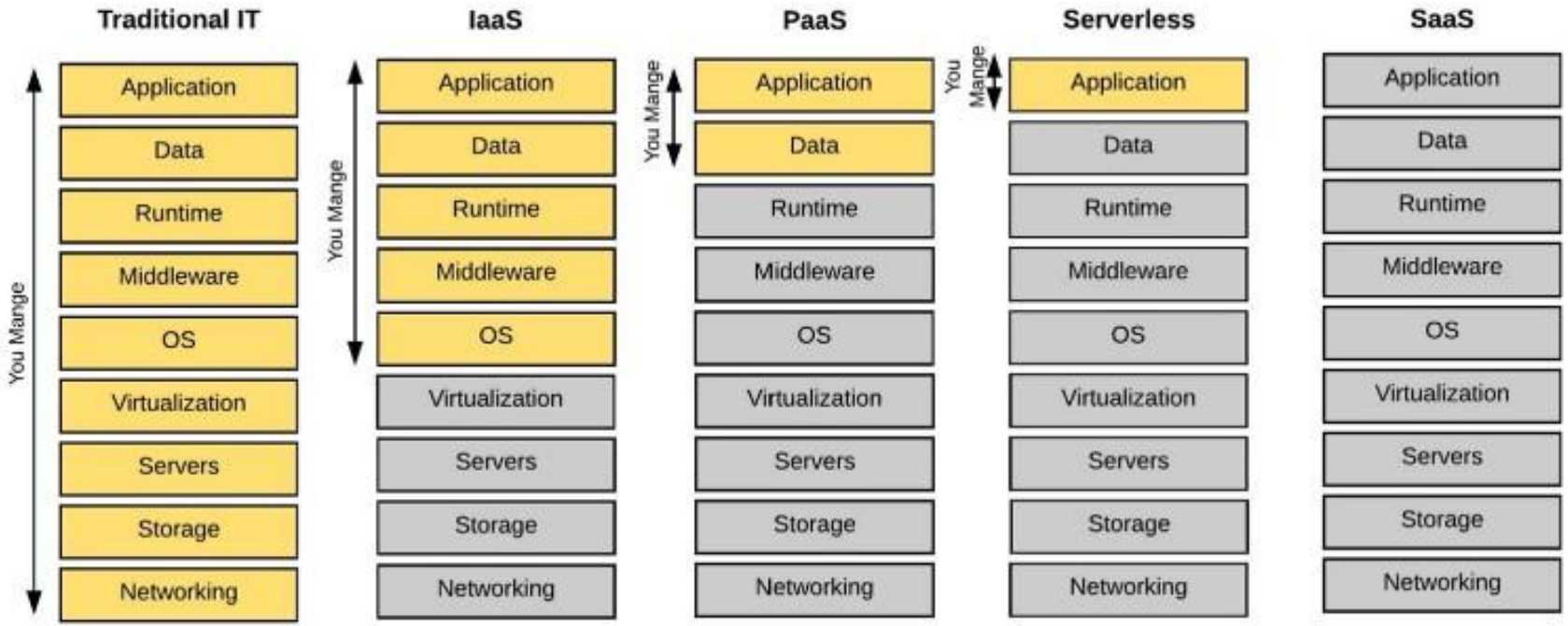
Matheus Rossato

- Sistemas de Informação e Administração na UFSC
- 12+ anos experiência em serviços de missão crítica
- Estagiário na Datasul CRM, Florianópolis/SC
- Responsável por Cloud na Chaordic, Florianópolis/SC
- Gerente de TI na ContaAzul, Joinville/SC

Evolução da computação

Para falar do futuro, primeiro precisamos entender o passado

Evolução da computação



TI Tradicional

- Servidores/datacenters físicos
- Prod, teste e dev acabavam diferentes
- Demora para deployar/iterar
- Único cliente
- Escalabilidade limitada, compra, entrega, configura, roda
- Não amigável com sistemas poliglotas, monoglota
- Deploy demora dias/semanas
- Executa durante meses/anos

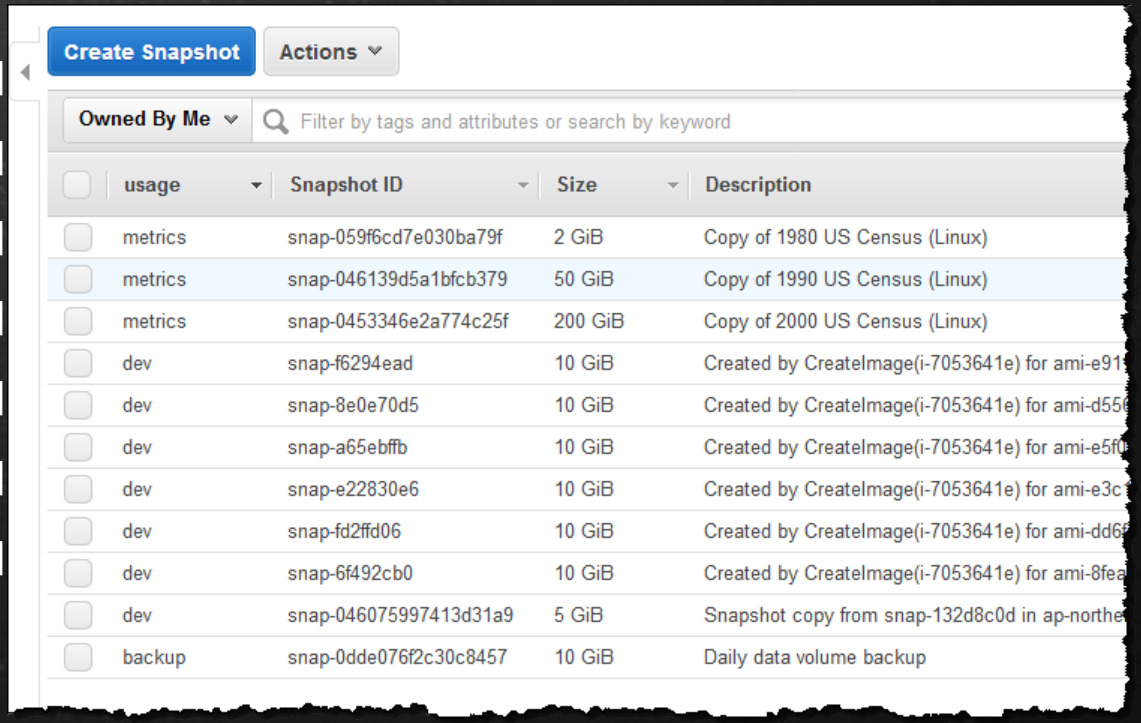
TI Tradicional

-
-
-
-
-
-

IaaS - Infrastructure as a Service

- Servidores virtuais
- Prod idealmente é imutável
- Rápido deploy/iteração
- Multi-clientes
- Amigável com sistemas poliglotas
- Deploy demora minutos/horas
- Executa por semanas/meses

IaaS - Infrastructure as a Service



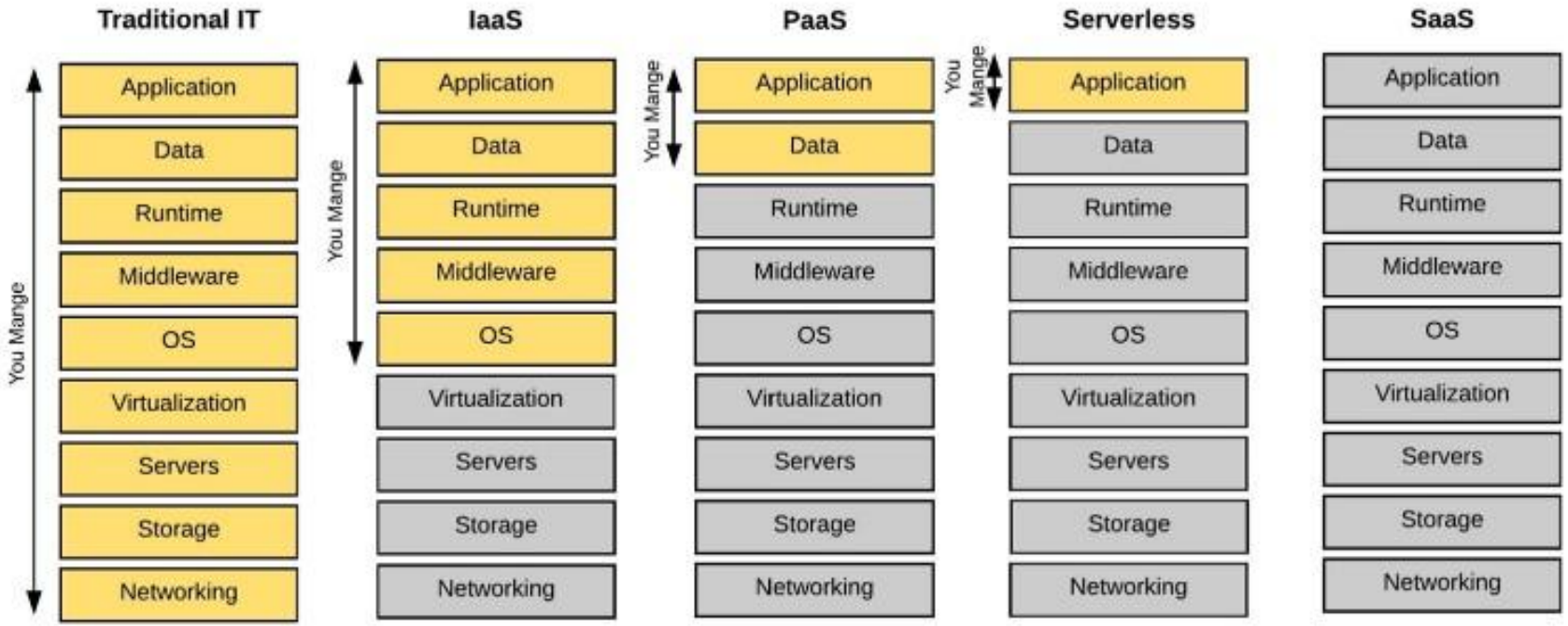
PaaS - Platform as a Service

- Containers
- Test e prod são idênticos, empurra o container para prod
- Muito rápido deploy/iteração
- Grande densidade de multi-clientes
- Amigável com sistemas poliglotas
- Deploy demora segundos/minutos
- Executa por horas/semanas

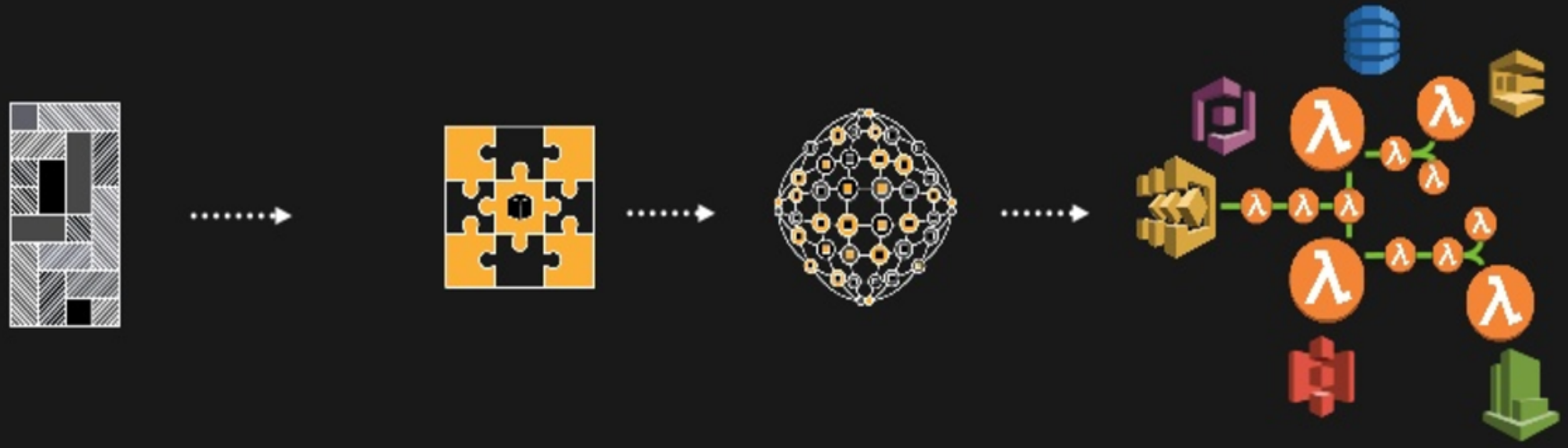
FaaS - Functions as a Service

- Serverless
- Menor unidade gerenciável do sistema que entrega valor
- Test e prod são os mesmos, sem acesso ao ambiente
- Muito rápido deploy/iteração
- Extrema densidade de multi-clientes
- Muito amigável com sistemas poliglotas
- Deploy demora segundos/minutos
- Executa por milisegundos/segundos/minutos
- BaaS, Backend as a Service

Evolução da computação



Evolução arquitetural para Serverless



Monolithic Application

Services

Microservices

Serverless

Computação Serverless - Escala

- IaaS = VMs
  - Máquina Virtual como unidade de escala
  - Abstraia o hardware
- PaaS = Containers
  - Aplicação como unidade de escala
  - Abstraia o sistema operacional
- FaaS = Funções Serverless
  - Função como unidade de escala
  - Abstraia o runtime da linguagem

Serverless

Já deu para entender o que é serverless afinal?

- Ainda existem servidores, mas não são mais geridos por você
- Quer dizer que você não tem mais acesso a eles também
- Dessa forma, não precisa gerir/otimizar eles
- Serverless = Microservices - Gerenciamento

AWS Lambda

Suporta:

- ❑Node.js
- ❑Java 8
- ❑C#
- ❑Python

Roda:

- ❑PHP, Ruby, Go

Frameworks:

- ❑Serverless = Todas suportadas
- ❑Apex = Todas - C# + Go + Closure
- ❑Chalice = Python

Benefícios

- Pago apenas pelo uso
- Não paga por recurso ocioso
- Cobra por 100s milisegundos e não por hora
- Escala de acordo com o uso
- Alta disponibilidade e tolerância a falha incluso
- Segurança, rodar por alguns segundos reduz riscos
- Foco no desenvolvimento e entrega de valor para o cliente

Pontos de atenção

- Tempo de execução, máx 5min
- Tempo de inicialização, médio de 10s
- Testes de integração ficam mais difíceis
- Deploy, empacotamento e versionamento sem padrão definido
- Descoberta de funções ainda não endereçado
- Tamanho das filas
- Persistência de dados
- Degradação graciosa e fallback
- Ignorar operação
    - Serverless != No Ops

The Serverless Ecosystem


Amazon Lex


Amazon Glacier


AWS Lambda


Amazon Cognito


Amazon Kinesis


AWS Step Functions


Amazon SQS


Amazon Pinpoint


Amazon AppStream


Amazon Polly


Amazon SNS


Amazon S3


Amazon Rekognition


Amazon Athena

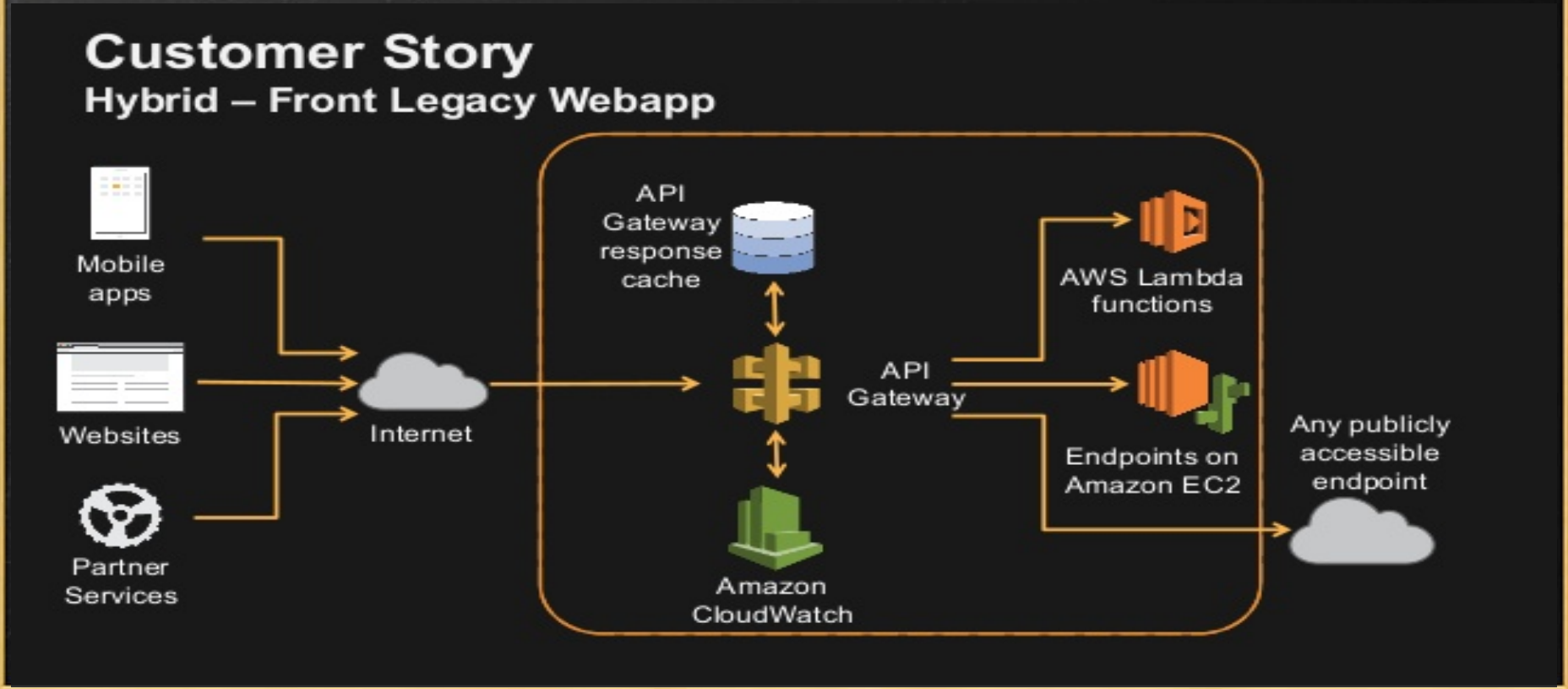

Amazon API Gateway


Amazon CloudWatch


AWS IoT


Mobile Analytics

Exemplo de arquitetura Serverless



Até agora

Custo e risco diminuem

Ritmo de mudança e complexidade aumentam

Site Reliability Engineering

Termo cunhado pelo Google, que define como:

"Quando se contrata Engenheiros de Software para rodar os produtos e criar sistemas que executam o trabalho, que de outra forma seria executado muitas vezes, manualmente, por administradores de sistemas"

Site Reliability Engineering

Os SREs tem como foco, os aspectos abaixo, sob seus serviços:

- ❑disponibilidade
- ❑latência
- ❑performance
- ❑eficiência
- ❑gestão de mudança
- ❑monitoração
- ❑resposta a incidentes
- ❑planejamento de capacidade

- ❑ ? segurança ? = negócio/produto

Site Reliability Engineering

Em essência:

- Limita o tempo dos desenvolvedores em 50% de operação
- Excedente transborda para o time de produto
- Meta de no máximo 2 incidentes por turno/dia
- PostMortem para todos incidentes graves = afetam usuários
- Garanta que o PostMortem seja Blameless = sem buscar culpados, mas sim como evitar que ocorra novamente
  - Senão começa uma atitude de "cover your ass"
- Plano de Ação com prioridades, alinhado com executivos
  - Prioridade não tem plural, ex: 3 prioridades 1 = nenhuma

SRE

- Não focar em 100% de disponibilidade, pois é ineficiente
- Negócio define o nível desejado, senão cliente pode ir embora
- Medir o risco de indisponibilidade não planejada
- Baseado em Tempo

=> Disponibilidade = Uptime/(Uptime+Downtime)

- Baseado em Requisições

=> Disponibilidade = Requisições OK/Total de Requisições

SRE

Nível de Serviço

SLA - Service Level Agreement
- Negócio e produto definem, em geral associado a multa

SLI - Service Level Indicator
- Métrica usada para acompanhar o SLO
- Use Percentil e não média, analisando assim os casos piores

SLO - Service Lever Objective
- Meta que se pretende alcancar na operação do serviço

SRE

Nível de Serviço e monitoração

- Muitos indicadores = difícil prestar atenção
- Poucos indicadores = talvez ignore comportamento relevante
- Diferente classes de serviços = indicadores diferentes
  - User-facing: disponibilidade, latência, vazão/requisições
  - Armazenamento: latência, disponibilidade, durabilidade
  - BigData: vazão/requisições, latência do fluxo
  - Todos sistemas se importam com dados corretos

SRE

Monitoração

- Importante para:
  - Tendências de longo prazo
  - Comparação e execução de experimentos
- Não deve ser necessário um humano interpretar a mensagem
  - Evitar sistemas mágicos, em geral mais trabalho
- 3 saídas da monitoração:
  - Alertas = ação necessária imediata por humano
  - Tickets = ação eventual de um humano
  - Logs = sem ação necessária

SRE

Gestão de mudança

- 70% dos incidentes são devido a liberação de versão
  - Remova o humano do processo, repetição gera negligência
- Implemente liberação progressiva
  - Deploy de Canário
  - Azul/Verde
  - Rollback automático > investigar com cliente impactado
- Monitoração eficiente importante

SRE

Resposta a incidente

- Função entre MTTF e MTTR
  - MTTF = Mean Time to Failure
  - MTTR = Mean Time to Recover
- Humanos adicionam latência ao MTTR
  - Sistemas que se recuperam sem intervenção humana vão ter menor MTTR, portanto maior disponibilidade
- Playbooks > Heróis (3x)
  - Treinamento de incidente ajuda a validar playbooks

E: matheusrossato@gmail.com
L:/in/matheusrossato

Referências

Diagrama Evolução Computação
https://medium.com/@sdorzak/why-serverless-is-the-new-black-e4ff9e9947e0

Serverless Ecosystem
https://pt.slideshare.net/AmazonWebServices/a-brief-look-at-serverless-architecture

Martin Fowler
https://martinfowler.com/articles/serverless.html

Getting Started with Serverless
https://www.youtube.com/watch?v=OI_V6OZZkZM&t=192s

Site Reliabilty Engineering
https://landing.google.com/sre/book/index.html