

```
{  
  "Version": "2021-03-25",  
  "Statement": [  
    {  
      "Sid": "Serverless:está:pronto?",  
      "Action": "Nós:Desenvolvedores:Não",  
      "Resource":  
        "arn:aws:Ibrahim-Cesar:sa-east-1"  
    }  
  ]  
}
```



@ibrahimcesar

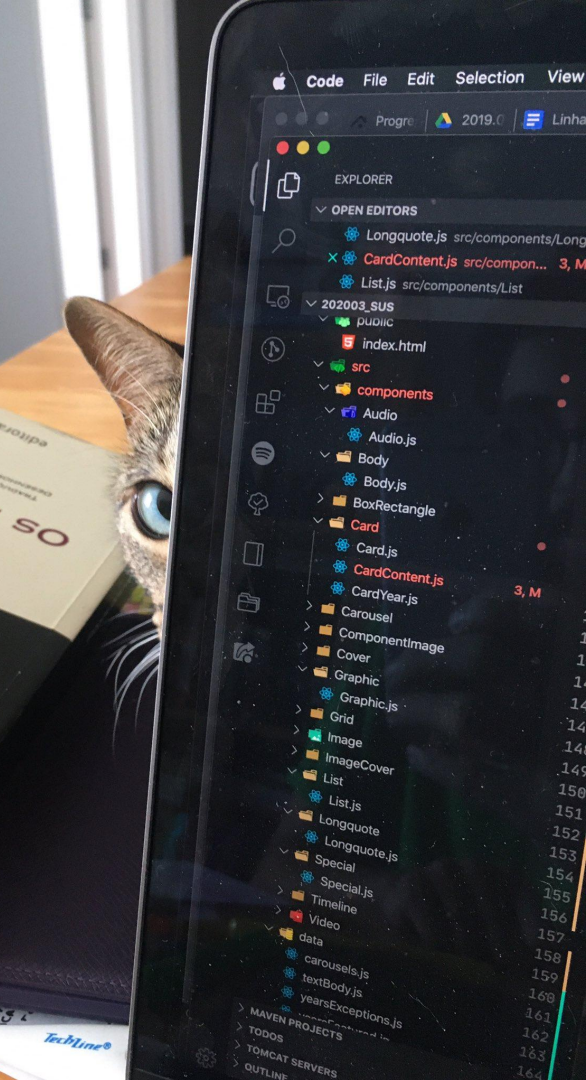
<https://ibrahimcesar.cloud>

```
{  
  "Version": "2021-03-25",  
  "Statement": [  
    {  
      "Sid": "Serverless:está:pronto?",  
      "Action": "Nós:Desenvolvedores:Não",  
      "Resource":  
        "arn:aws:Ibrahim-Cesar:sa-east-1"  
    }  
  ]  
}
```



@ibrahimcesar

<https://ibrahimcesar.cloud>



AWS community builder

Ibrahim Cesar Nogueira Bevilacqua

Ele/dele

Diretor de tecnologia **Nexo**

Nexo Jornal, Gama Revista e
Nexo Políticas Públicas

AWS Solutions Architect - Associate


AWS Certified Cloud Practitioner

Desenvolvedor web / cloud / serverless

@ibrahimcesar

<https://ibrahimcesar.cloud>



 @ibrahimcesar



Andrew Brown
@andrewbrown

I just drafted an **#AWS** Serverless Course Outline:
Tell me what you think so far.

[Traduzir Tweet](#)

... and runtime limitations made common tasks impossible
... of tolerate to write to integrate services
... my favourite web-frameworks and had to start from scratch
... impossible and painful to run or debug serverless services
... of work to write CloudFormation templates
... servability was extremely limited
... were too slow and hindered the UX
... serverless feasible relied heavily on expensive third-party providers
... ure gaps

Ready
that it is 2021 AWS has solved these previous issues and a
st approach is ready for any size of project or team.

re Not
effort to quickly upskill as a developer or a team still takes se
So it's no surprise that even though it's possible, teams are
... the old ways to meet their deadlines.

AWS Serverless Course Outline

Serverless is Ready: Developers Are Not The Obstacle

Serverless is Ready

I believe now that it is 2021 AWS has solved these previous issues and a
Serverless-first approach is ready for any size of project or team.

Developers are Not

However, the effort to quickly upskill as a developer or a team still takes several months
to even a year. So it's no surprise that even though it's possible, teams and developers
quickly go back to the old ways to meet their deadlines.

6:27 PM · 29 de dez de

Fundamentals of Software Architecture

An Engineering Approach

Mark Richards & Neal Ford

Everything in software architecture is a trade-off.

—First Law of Software Architecture

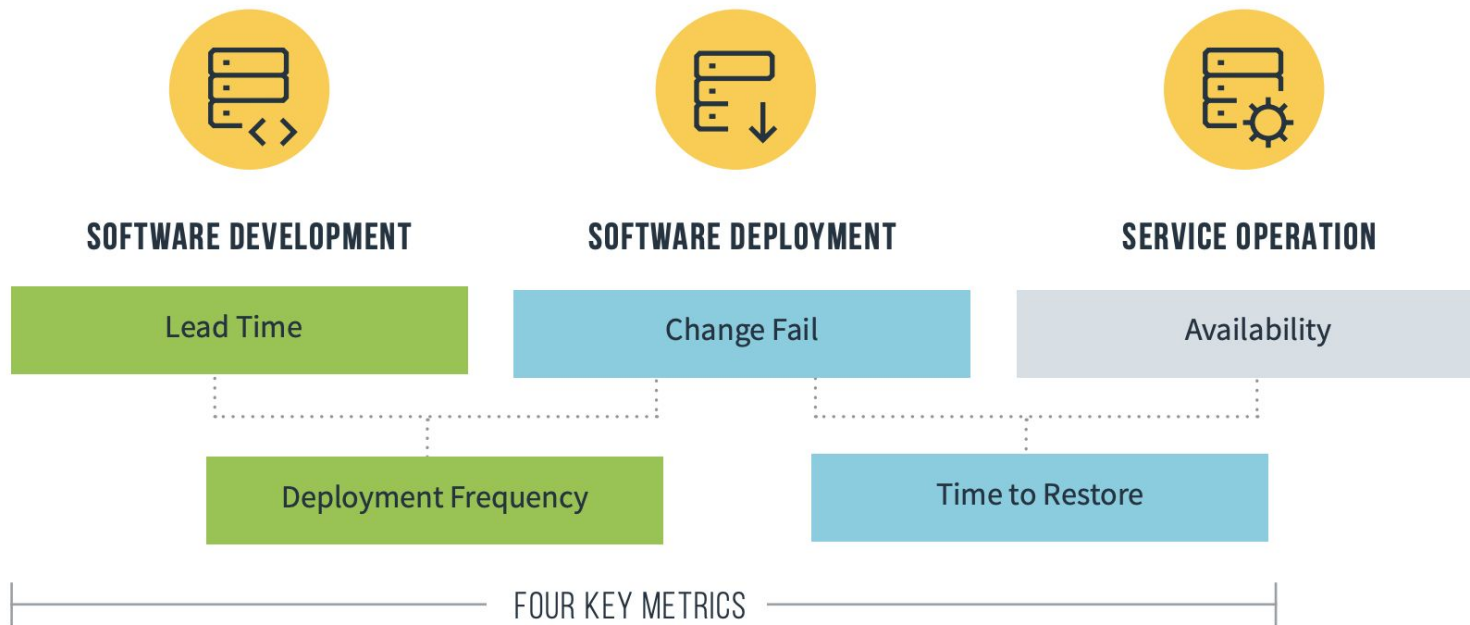
If an architect thinks they have discovered something that isn't a trade-off, more likely they just haven't identified the trade-off yet.

—Corollary 1

Why is more important than how.

—Second Law of Software Architecture

PERFORMANCE METRICS



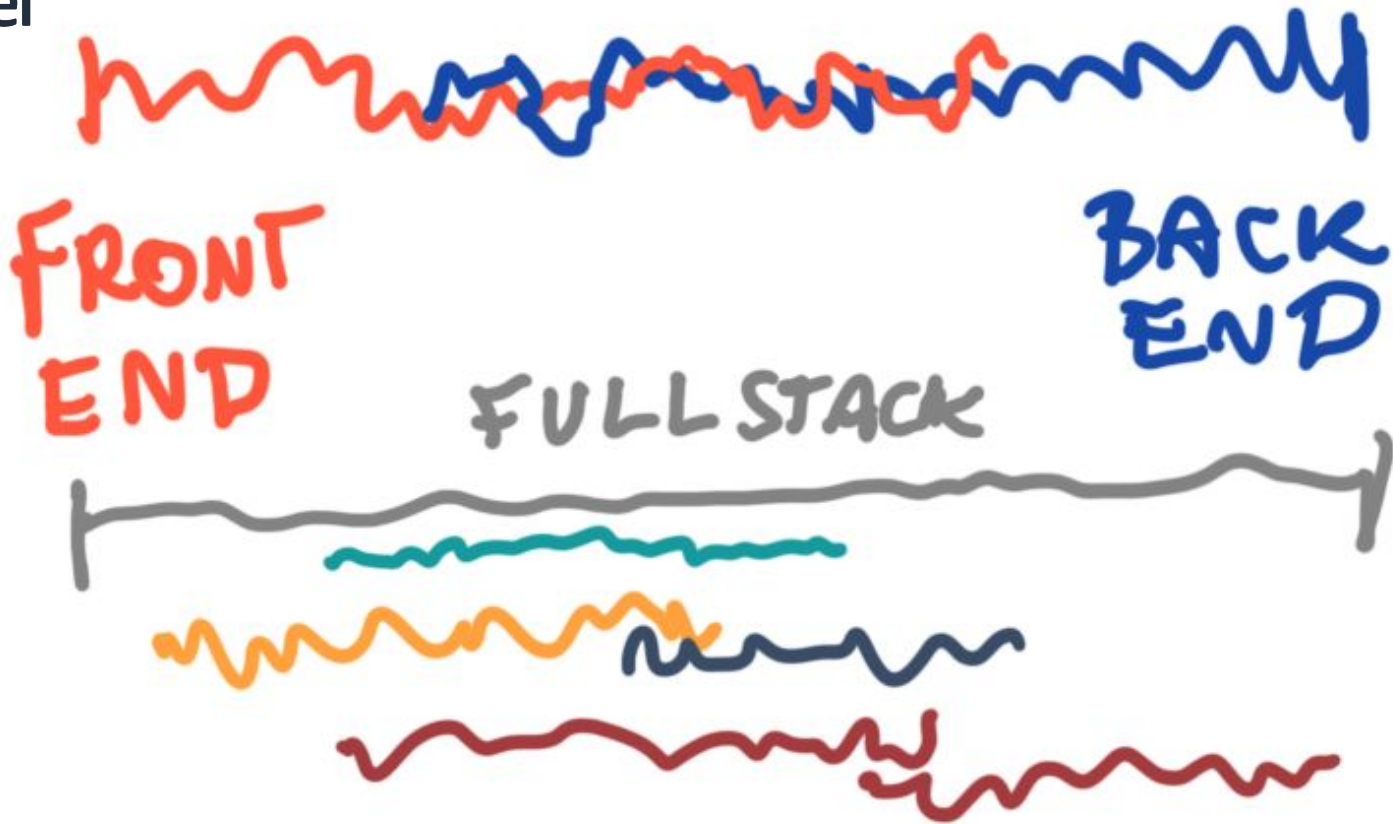
FRONTEND BACKEND





< O que é um mestre? Me surpreendo a todo momento com Python />

– Guido van Rossum, criador da linguagem Python



< Programadores conhecem os benefícios de tudo e os trade-offs de nada.

*Arquitetos precisam saber dos dois. />**

– Rich Hickey, criador da linguagem Clojure

** < Programmers know the the benefits of everything and the tradeoffs of nothing />*

{ <https://www.youtube.com/watch?v=rI8tNMsozo0&t=744s> }

Do que falamos quando falamos em serverless?

SERVER

Do que falamos quando falamos em serverless?

SERVER 

Do que falamos quando falamos em serverless?

SERVER

LESS

< aplicações sem servidor />

Do que falamos quando falamos em serverless?

SERVER
LESS

*< Serverless permite a criação e rodar aplicações e serviço **sem pensar a respeito de servidores** />**

– AWS

** < Serverless allows you to build and run applications and services without thinking about servers />*
<https://aws.amazon.com/serverless/>

Do que falamos quando falamos em serverless?

SERVER
LESS



2014-11-13

Lançada em Preview

FaaS

Function as a Service

* < Serverless allows you to build and run applications and services
without thinking about servers />
<https://aws.amazon.com/serverless/>

SERVER LESS

Do que falamos quando falamos em serverless?



*{ Metade das aplicações das novas
aplicações **são realizadas em lambdas** }*

– Andy Jessy, CEO AWS, re:Invent 2020

Do que falamos quando falamos em serverless?

Lambda oferece **funções como service**

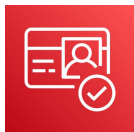


AWS Lambda

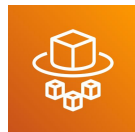
Serviços gerenciados oferecem **funcionalidades como service**



AWS Amplify



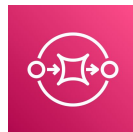
Amazon Cognito



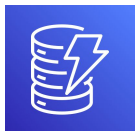
AWS Fargate



Amazon
Aurora
Serverless



SQS



Amazon
DynamoDB



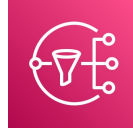
AWS Step
Functions



Amazon
EventBridge

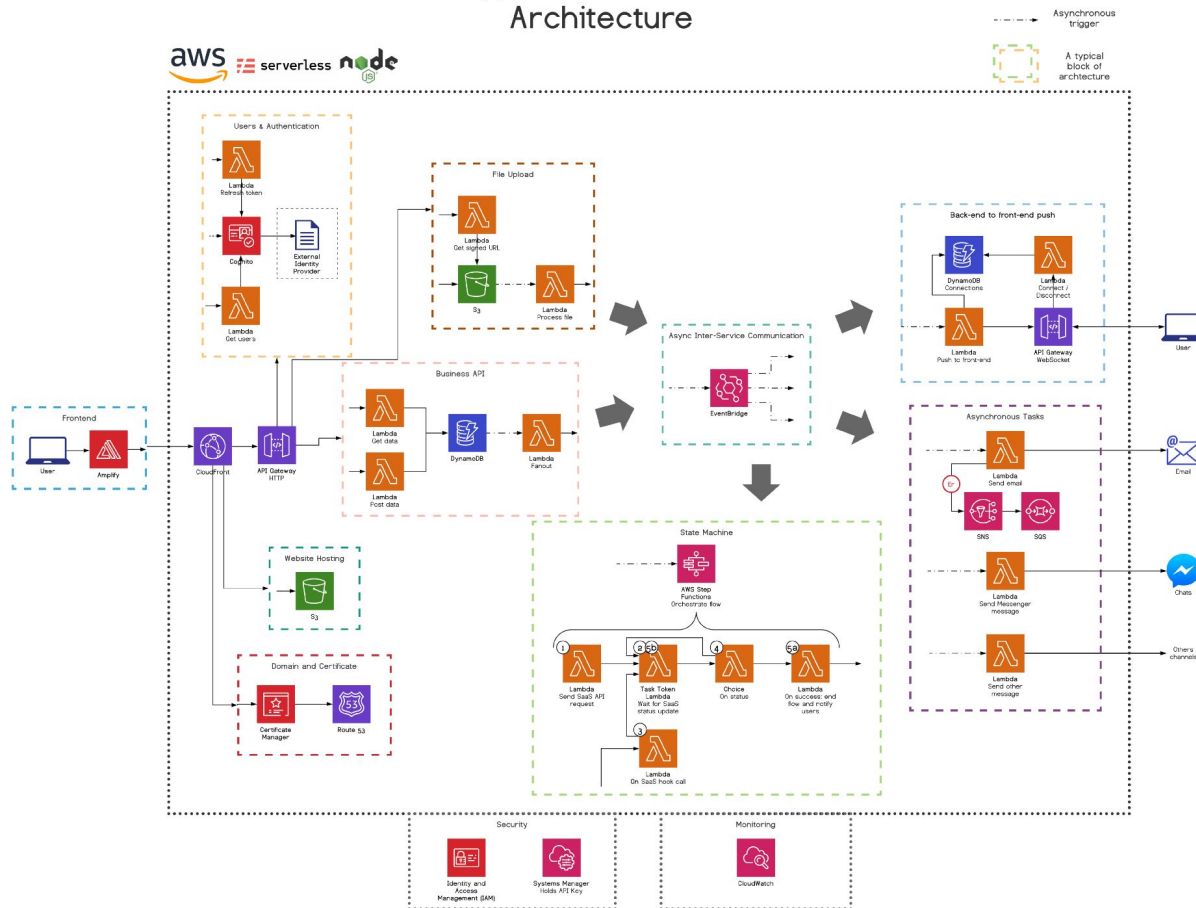


Amazon
API Gateway



SNS

Typical Serverless Architecture



Overview of Amazon Web Services

Publication date: **August 11, 2020** (*Document Details* (p. 65))

Abstract

Amazon Web Services offers a broad set of global cloud-based products including compute, storage, databases, analytics, networking, mobile, developer tools, management tools, IoT, security, and enterprise applications: on-demand, available in seconds, with pay-as-you-go pricing. From data warehousing to deployment tools, directories to content delivery, over 175 AWS services are available. New services can be provisioned quickly, without the upfront capital expense. This allows enterprises, start-ups, small and medium-sized businesses, and customers in the public sector to access the building blocks they need to respond quickly to changing business requirements. This whitepaper provides you with an overview of the benefits of the AWS Cloud and introduces you to the services that make up the platform.

Overview of Amazon Web Services

Publication date: August 11, 2020 (*Document Details* (p. 65))

Abstract

Amazon Web Services offers a broad set of global cloud-based products including compute, storage, databases, analytics, networking, mobile, developer tools, management tools, IoT, security, and enterprise applications: on-demand, available in seconds, with pay-as-you-go pricing. From data warehousing to deployment tools, directories to content delivery, over 175 AWS services are available. New services can be provisioned quickly, without the upfront capital expense. This allows enterprises, start-ups, small and medium-sized businesses, and customers in the public sector to access the building blocks they need to respond quickly to changing business requirements. This whitepaper provides you with an overview of the benefits of the AWS Cloud and introduces you to the services that make up the platform.

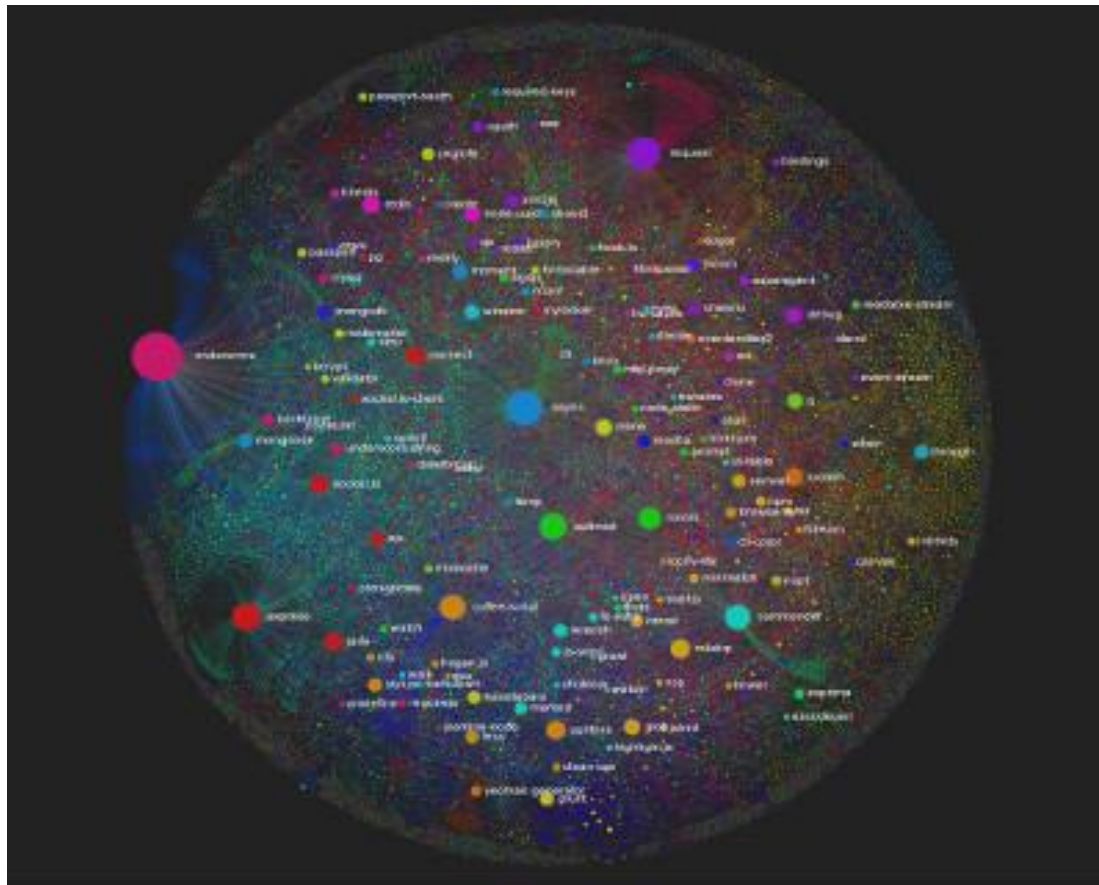
Overview of Amazon Web Services

Publication date: August 11, 2020 (*Document Details* (p. 65))

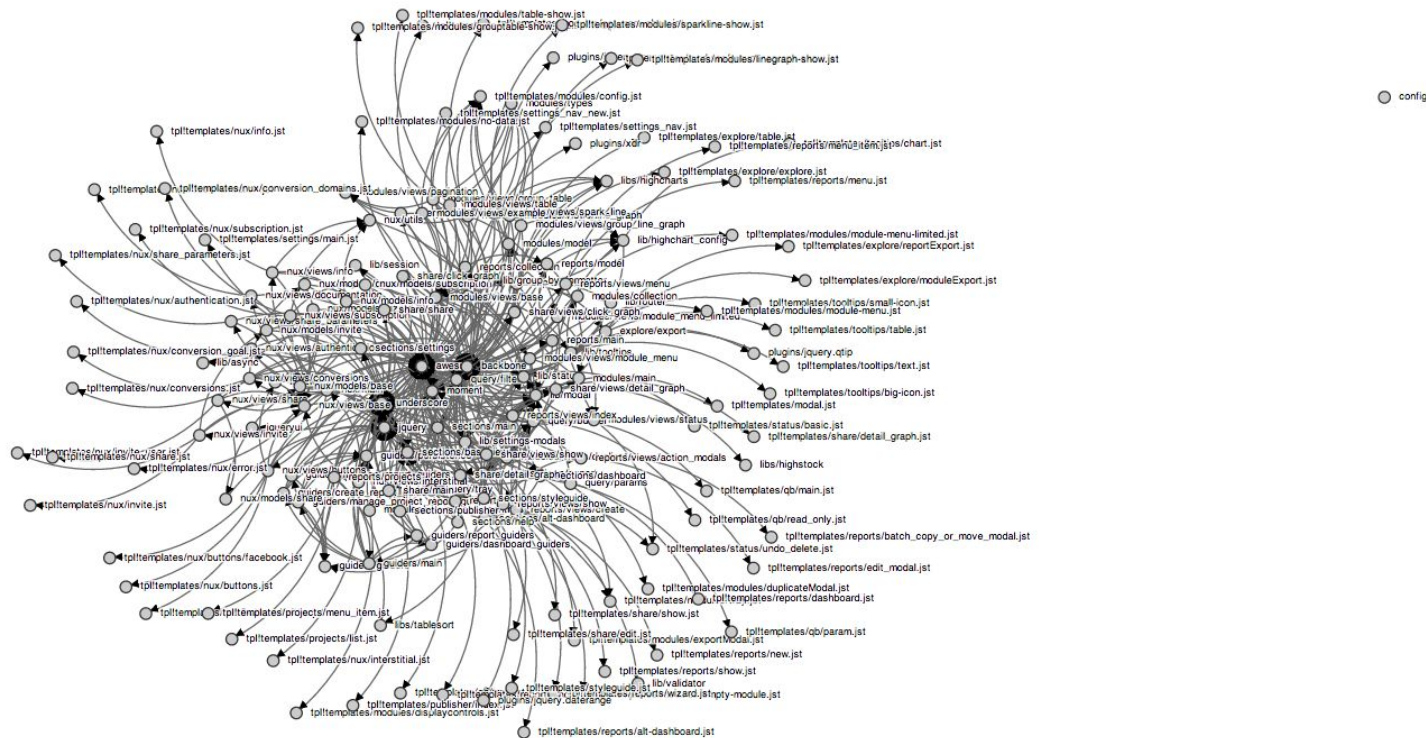
Abstract

Amazon Web Services offers a broad set of global cloud-based products including compute, storage, databases, analytics, networking, mobile, developer tools, management tools, IoT, security, and enterprise applications: on-demand, available in seconds, with pay-as-you-go pricing. From data warehousing to deployment tools, directories to content delivery, over 175 AWS services are available. New services can be provisioned quickly, without the upfront capital expense. This allows enterprises, start-ups, small and medium-sized businesses, and customers in the public sector to access the building blocks they need to respond quickly to changing business requirements. This whitepaper provides you with an overview of the benefits of the AWS Cloud and introduces you to the services that make up the platform.



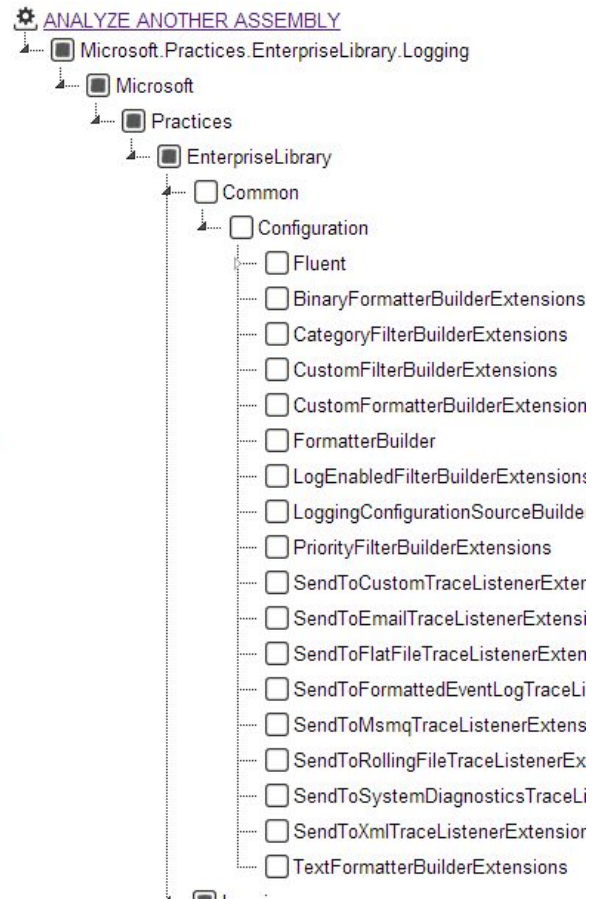


< Representação gráfica de dependências npm />



< Representação gráfica de dependências Ruby />

< Representação gráfica de dependências Python />



< Representação gráfica de dependências .NET />

HOW DO COMMITTEES INVENT?

by MELVIN E. CONWAY

That kind of intellectual activity which creates a useful whole from its diverse parts may be called the *design* of a *system*. Whether the particular activity is the creation of specifications for a major weapon system, the formation of a recommendation to meet a social challenge, or the programming of a computer, the general activity is largely the same.

Typically, the objective of a design organization is the creation and assembly of a document containing a coherently structured body of information. We may name this information the *system design*. It is typically produced for a sponsor who usually desires to carry out some activity guided by the system design. For example, a public official may wish to propose legislation to avert a recurrence of a recent disaster, so he appoints a team to explain the catastrophe. Or a manufacturer needs a new product and designates a product planning activity to specify what should be introduced.

The design organization may or may not be involved in the construction of the system it designs. Frequently, in public affairs, there are policies which discourage a group's acting upon its own recommendations, whereas, in private industry, quite the opposite situation often prevails.

It seems reasonable to suppose that the knowledge that one will have to carry out one's own recommendations or that this task will fall to others, probably affects some design choices which the individual designer is called upon to make. Most design activity requires continually making choices. Many of these choices may be more than design decisions; they may also be personal decisions the designer makes about his own future. As we shall see later, the incentives which exist in a conventional management environment can motivate choices which subvert the intent of the sponsor.

design organization criteria

ing a design team means that certain design decisions have already been made, explicitly or otherwise. Given any design team organization, there is a class of design alternatives which cannot be effectively pursued by such an organization because the necessary communication paths do not exist. Therefore, there is no such thing as a design group which is both organized and unbiased.

Once the organization of the design team is chosen, it is possible to delegate activities to the subgroups of the organization. Every time a delegation is made and somebody's scope of inquiry is narrowed, the class of design alternatives which can be effectively pursued is also narrowed.

Once scopes of activity are defined, a coordination problem is created. Coordination among task groups, although it appears to lower the productivity of the individual in the small group, provides the only possibility that the separate task groups will be able to consolidate their efforts into a unified system design.

Thus the life cycle of a system design effort proceeds through the following general stages:

1. Drawing of boundaries according to the ground rules.
2. Choice of a preliminary system concept.
3. Organization of the design activity and delegation of tasks according to that concept.
4. Coordination among delegated tasks.
5. Consolidation of subdesigns into a single design.

It is possible that a given design activity will not proceed straight through this list. It might conceivably reorganize upon discovery of a new, and obviously superior, design concept; but such an appearance of uncertainty is unflattering, and the very act of voluntarily abandoning a creation is painful and expensive. Of course, from the



{ 1968-04 }

HOW DO COMMITTEES INVENT?

by MELVIN E. CONWAY

That kind of intellectual activity which creates a useful whole from its diverse parts may be called the *design* of a *system*. Whether the particular activity is the creation of specifications for a major weapon system, the formation of a recommendation to meet a social challenge, or the programming of a computer, the general activity is largely the same.

Typically, the objective of a design organization is the creation and assembly of a document containing a coherently structured body of information. We may name this information the *system design*. It is typically produced for a sponsor who usually desires to carry out some activity guided by the system design. For example, a public official may wish to propose legislation to avert a recurrence of a recent disaster, so he appoints a team to explain the catastrophe. Or a manufacturer needs a new product and designates a product planning activity to specify what should be introduced.

The design organization may or may not be involved in the construction of the system it designs. Frequently, in public affairs, there are policies which discourage a group's acting upon its own recommendations, whereas, in private industry, quite the opposite situation often prevails.

It seems reasonable to suppose that the knowledge that one will have to carry out one's own recommendations or that this task will fall to others, probably affects some design choices which the individual designer is called upon to make. Most design activity requires continually making choices. Many of these choices may be more than design decisions; they may also be personal decisions the designer makes about his own future. As we shall see later, the incentives which exist in a conventional management environment can motivate choices which subvert the intent of the sponsor.

design organization criteria

design group and the system are identical. In the case where some group designed more than one subsystem we find that the structure of the design organization is a collapsed version of the structure of the system, with the subsystems having the same design group collapsing into one node representing that group.

This kind of a structure-preserving relationship between two sets of things is called a *homomorphism*. Speaking as a mathematician might, we would say that there is a homomorphism from the linear graph of a system to the linear graph of its design organization.

systems image their design groups

It is an article of faith among experienced system designers that given any system design, someone someday will find a better one to do the same job. In other words, it is misleading and incorrect to speak of *the* design for a specific job, unless this is understood in the context of space, time, knowledge, and technology. The humility which this belief should impose on system designers is the only appropriate posture for those who read history or consult their memories.

The design progress of computer translators of programming languages such as FORTRAN and COBOL is a case in

* This claim may be viewed several ways. It may be trivial, hinging on the definition of meaningful negotiation. Or, it may be the result of the observation that one design group almost never will compromise its own design to meet the needs of another group unless absolutely imperative.

30

4. Coordination among delegated tasks.
5. Consolidation of subdesigns into a single design.

It is possible that a given design activity will not proceed straight through this list. It might conceivably reorganize upon discovery of a new, and obviously superior, design concept; but such an appearance of uncertainty is unflattering, and the very act of voluntarily abandoning a creation is painful and expensive. Of course, from the

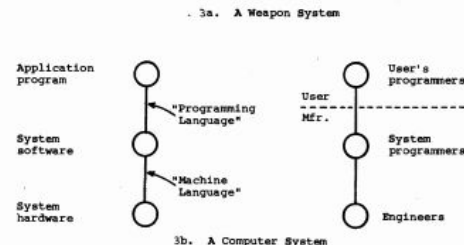


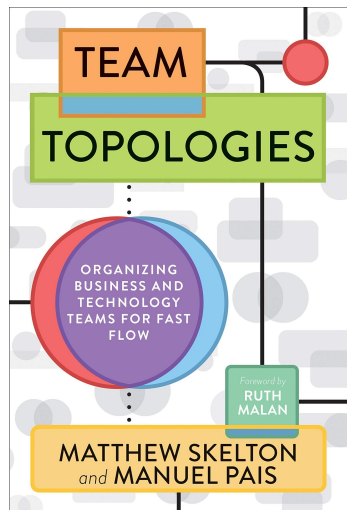
Figure 3 Two examples of identity of structure between a system and its design organization.

Figs. 3a and 3b

respective needs. After great effort they produced a copy of their organization chart. (See Fig. 3a.)

Consider the operating computer system in use solving a problem. At a high level of examination, it consists of three parts: the hardware, the system software, and the application program. (See Fig. 3b.) Corresponding to these subsystems are their respective designers: the computer manufacturer's engineers, his system programmers, and the user's application programmers. (Those rare instances where the system hardware and software tend to cooperate rather than merely tolerate each other are associated with

DATAMATION



< Team Topologies

– Matthew Skelton e Manuel Pais />





Combine sua arquitetura organizacional com sua arquitetura de software

{

*“Esqueça monolitos vs. microserviços.
Esforço cognitivo é o que importa”*

}

4 fundamental topologies

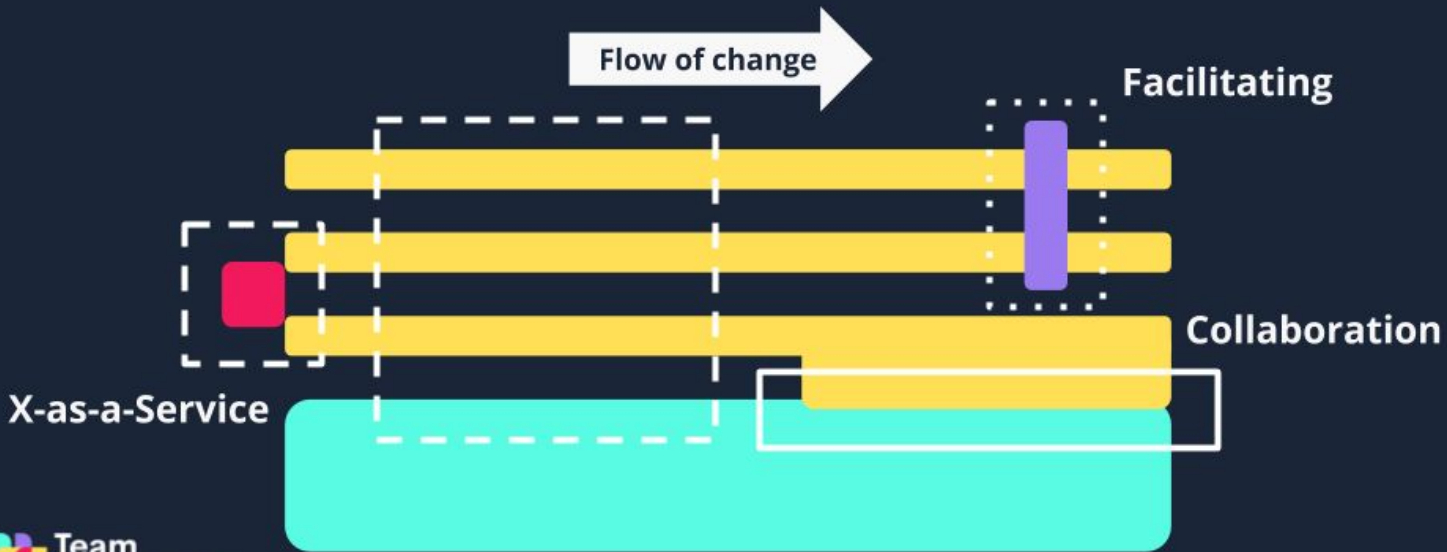
-  Stream-aligned team
-  Enabling team
-  Complicated Subsystem team
-  Platform team

4 fundamental topologies

Flow of change



3 core interaction modes





< Intro de Esforço Cognitivo/>
COGNITIVE
LOAD 101

Três tipos de esforço cognitivo

Esforço intrínseco < *Intrinsic cognitive load* />

{ Se relaciona com aspectos fundamentais do espaço do problema.

Exemplos: Como se define uma classe em Java?

O que é o serviço X? }

Esforço estranho

{ Se relaciona com aspectos do ambiente ou tarefas repetitivas ou com incertezas.

Exemplos: Como se faz mesmo o deploy?

Como eu inicializo um aplicação no Amplify? }

Esforço pertinente

{ Se relaciona com aspectos das tarefas que precisam de atenção especial, para aprendizado e alta performance

Exemplos: Como integrar DynamoDB streams com esta função Lambda? }

Três tipos de esforço cognitivo

Esforço intrínseco *< Intrinsic cognitive load />*

{ Se relaciona com aspectos fundamentais do espaço do problema.

Exemplos: Como se define uma classe em Java?

O que é o serviço X? }



< Deve ser **minimizado** – através de treinamento, escolhas acertadas de tecnologia para as habilidades do time, contratação, arquitetura, programação em par, *mob programming* >

s repetitivas ou com incertezas.

o Amplify? }

Esforço pertinente

{ Se relaciona com aspectos das tarefas que precisam de atenção especial, para aprendizado e alta performance

Exemplos: Como integrar DynamoDB streams com esta função Lambda? }

Três tipos de esforço cognitivo

Esforço intrínseco

{ Se relaciona com aspectos fundamentais do espaço do problema.

Exemplos: Como se define uma classe em Java?

O que é o serviço X? }

Esforço estranho *< Extraneous cognitive load />*

{ Se relaciona com aspectos do ambiente ou tarefas repetitivas ou com incertezas.

Exemplos: Como se faz mesmo o deploy?

Como eu inicializo um aplicação? }

Esforço pertinente

{ Se relaciona com aspectos das tarefas que precisam de atenção especial, para aprendizado e alta performance

Exemplos: Como integrar DynamoDB streams com esta função Lambda? }

Três tipos de esforço cognitivo

< Deve ser **eliminado** – através de uma cultura de DevOps, automatização de boas práticas de segurança, eliminar comunicações desnecessárias, processos, *runbooks*, *playbooks*, ferramentas externas, ***uso de serviços gerenciados*** >

ação do problema.
ava?

Esforço estranho *< Extraneous cognitive load />*

- { Se relaciona com aspectos do ambiente ou tarefas repetitivas ou com incertezas.
Exemplos: Como se faz mesmo o deploy?
Como eu inicializo um aplicação? }

Esforço pertinente

- { Se relaciona com aspectos das tarefas que precisam de atenção especial, para aprendizado e alta performance
Exemplos: Como integrar DynamoDB streams com esta função Lambda? }

Três tipos de esforço cognitivo

Esforço intrínseco

{ Se relaciona com aspectos fundamentais do espaço do problema.

Exemplos: Como se define uma classe em Java?

O que é o serviço X? }

Esforço estranho

{ Se relaciona com aspectos do ambiente ou tarefas repetitivas ou com incertezas.

Exemplos: Como se faz mesmo o deploy?

Como eu inicializo um aplicação? }

Esforço pertinente *< Germane cognitive load />*

{ Se relaciona com aspectos das tarefas que precisam de atenção especial, para aprendizado e alta performance

Exemplos: Como integrar DynamoDB streams com esta função Lambda? }

Três tipos de esforço cognitivo

Esforço intrínseco

{ Se relaciona com aspectos fundamentais do espaço do problema.

Exemplos: Como se define uma classe em Java?

O que é o serviço X? }

Esforço estranho

{ < É aqui que ocorre a produtividade, escrita de código que adiciona valor ao negócio, concentração em tarefas desnecessárias >

E

S

load />



{ Se relaciona com aspectos das tarefas que precisam de atenção especial, para aprendizado e alta performance

Exemplos: Como integrar DynamoDB streams com esta função Lambda? }

*< Cada linha de código é uma
decisão de compra. />**

– Bill Buckley

** < Every line of code is a buying decision />*

*As relações humano-tecnológicas são uma dança sutil nas quais os objetos tecnológicos **empurram e puxam com vários degraus de insistência** enquanto os humanos navegam com **motivação, criatividade e habilidade em diferentes níveis**.*

Tecnologias são criadas, implementadas e utilizadas através de uma rede de escolhas.

– How Artifacts Afford, Jenny L. Davis

Full Stack Serverless, Cloud Engineers, Cloud Developers, Cloud Ops, Cloud <...> devem ser cada vez **mais arquitetos**, entender mais os **trade-offs**, a complexidade da Cloud só torna **menos opaca as abstrações** e os **primitivos** para criação de nossas aplicações.

A Cloud e o serverless estão maduros.
Precisamos fazer **decisões informadas** e explorar esse imenso potencial e fazer **escolhas explícitas e bem informadas**, respeitando a topologia de nossos times.

<Obrigado />



@ibrahimcesar
<https://ibrahimcesar.cloud>

AWS
community
builder