Architecture as Code (AaC) with Python

Ruslan Korniichuk | 11.12.2024

TABLE OF CONTENTS

01

Introduction

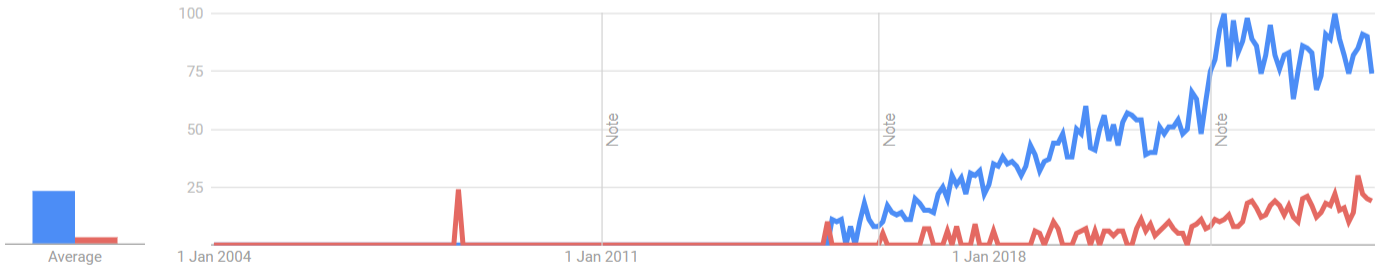
Infrastructure as Code
Search term

Architecture as Code
Search term

+ Add comparison

United States | 2004 – present | All categories | Web Search

Interest over time



Compared breakdown by sub-region

Sub-region

Infrastructure as ...
Search term

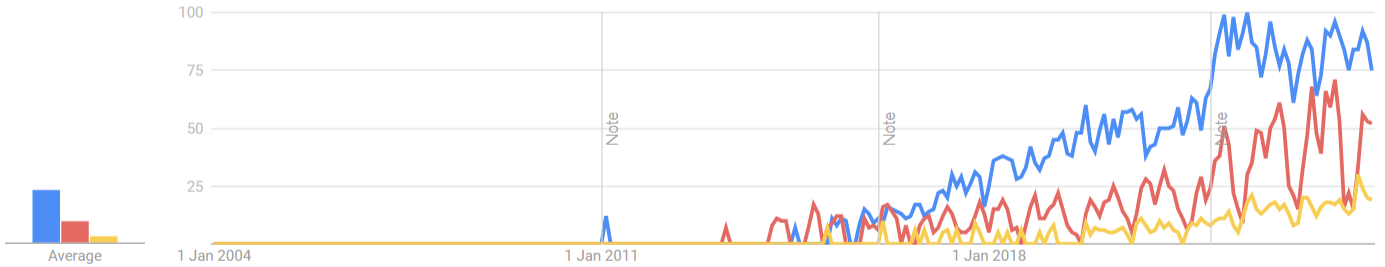
Diagram as Code
Search term

Architecture as C...
Search term

+ Add comparison

United States | 2004 – present | All categories | Web Search

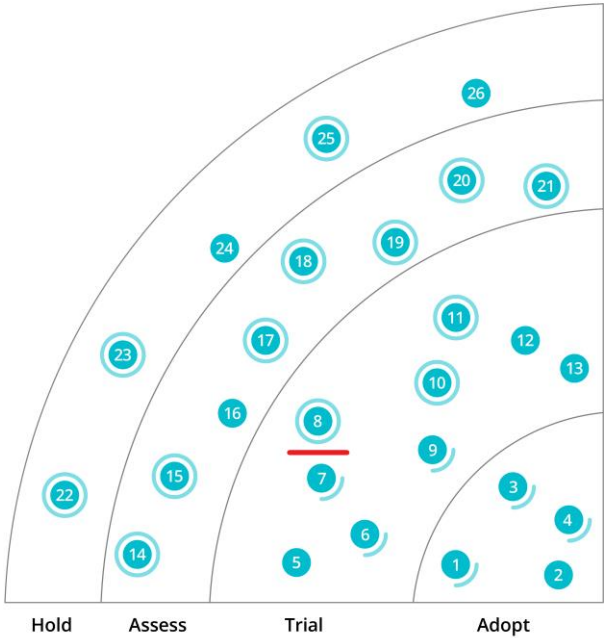
Interest over time



Compared breakdown by sub-region

Sub-region

Technology Radar by ThoughtWorks



Adopt
1. Dependency drift fitness function
2. Run cost as architecture fitness function
3. Security policy as code
4. Tailored service templates

Trial
5. Continuous delivery for machine learning (CD4ML)
6. Data mesh
7. Declarative data pipeline definition
8. Diagrams as code
9. Distroless Docker images
10. Event interception
11. Parallel run with reconciliation
12. Use "remote native" processes and approaches
13. Zero trust architecture

Assess
14. Bounded low-code platforms
15. Browser-tailored polyfills
16. Decentralized identity
17. Kube-managed cloud services
18. Open Application Model (OAM)
19. Secure enclaves
20. Switchback experimentation
21. Verifiable credentials

Hold
22. Apollo Federation
23. ESBs in API Gateway's clothing
24. Log aggregation for business analytics
25. Micro frontend anarchy
26. Productionizing notebooks

Source: tinyurl.com/2nrh4uw2

Infrastructure as Code vs Diagram as Code vs Architecture as Code

Infrastructure as Code (IaC) is the managing and provisioning of infrastructure through code.
Tools: Terraform, Pulumi, AWS Cloud Development Kit (CDK).

Infrastructure as Code vs Diagram as Code vs Architecture as Code

Infrastructure as Code (IaC) is the managing and provisioning of infrastructure through code.
Tools: Terraform, Pulumi, AWS Cloud Development Kit (CDK).

Diagram as Code is the creation of diagrams using code.
Tools: Diagrams (Python), Ilograph, Mermaid, Structurizr DSL.

Infrastructure as Code vs Diagram as Code vs Architecture as Code

Infrastructure as Code (IaC) is the managing and provisioning of infrastructure through code.
Tools: Terraform, Pulumi, AWS Cloud Development Kit (CDK).

Diagram as Code is the creation of diagrams using code.
Tools: Diagrams (Python), Ilograph, Mermaid, Structurizr DSL.

Architecture as Code (AaC) is the prototyping and visualization of system architecture using code.
Tools: Diagrams (Python).

Diagram as Code tools

- AsciiDoctor Diagram: set of Asciidoctor extensions
- CloudGram: Domain-Specific Language (DSL)
- D2: DSL
- Graphviz: DOT language
- Ilograph: YAML language
- Mermaid: Mermaid's syntax (Markdown-inspired)
- PlantUML: PlantUML language
- Structurizr DSL: DSL
- WebSequenceDiagrams: DSL

Static vs interactive

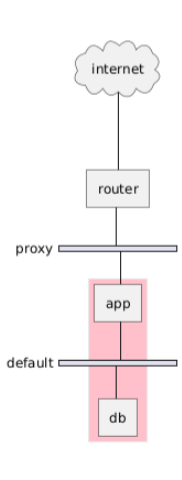
Static diagrams (e.g., PNG, SVG):

- D2
- Graphviz
- Mermaid
- PlantUML
- WebSequenceDiagrams

Interactive diagrams (e.g., browser-based):

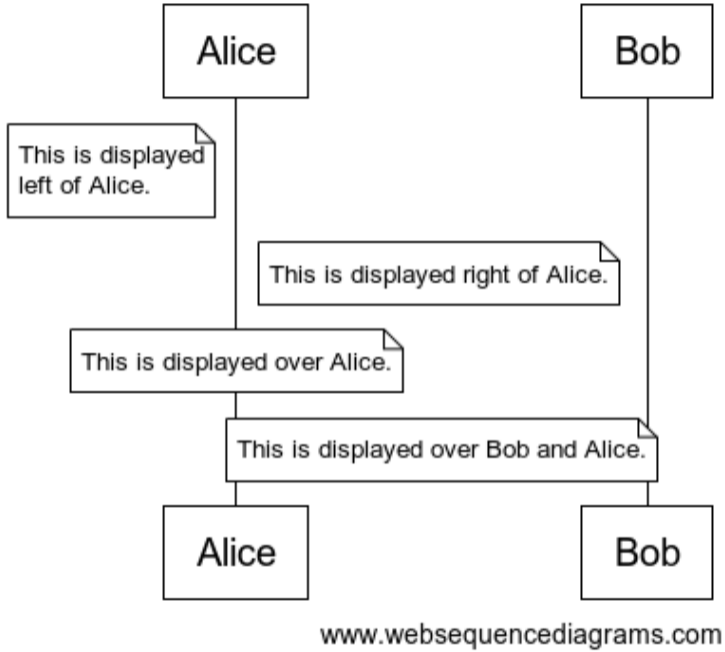
- CloudGram
- Ilograph
- Structurizr DSL

```
@startuml
nwdiag {
    internet [ shape = cloud];
    internet -- router;

    network proxy {
        router;
        app;
    }
    network default {
        app;
        db;
    }
    group {
      color = "pink";
      app;
      db;
    }
}
@enduml
```

File Edit View Style Help Year end sale Sign in

```
participant Alice
participant Bob

note left of Alice
This is displayed
left of Alice.
end note
note right of Alice: This is displayed right of Alice.
note over Alice: This is displayed over Alice.
note over Alice, Bob: This is displayed over Bob and Alice.
```



Untitled (unsaved)

There are no errors in the syntax of the diagram.

Ln 10, Col 60 0

Mermaid Live Editor Playground

v11.4.0 SECURITY THEME DOCUMENTATION

Mermaid Code Config Auto sync DOCS

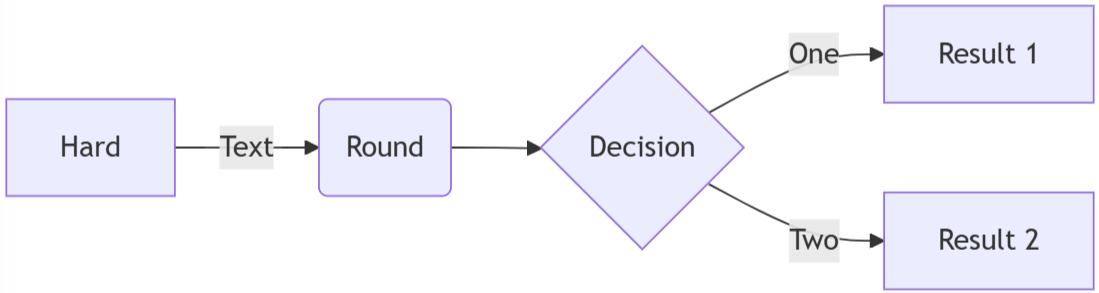
```
flowchart LR
    A[Hard] -->|Text| B(Round)
    B --> C{Decision}
    C -->|One| D[Result 1]
    C -->|Two| E[Result 2]

```

Sample Diagrams

History

Actions

Diagram Rough Pan & Zoom FULL SCREEN SAVE TO MERMAID CHART



Mermaid | Code | Config | Auto sync | DOCS

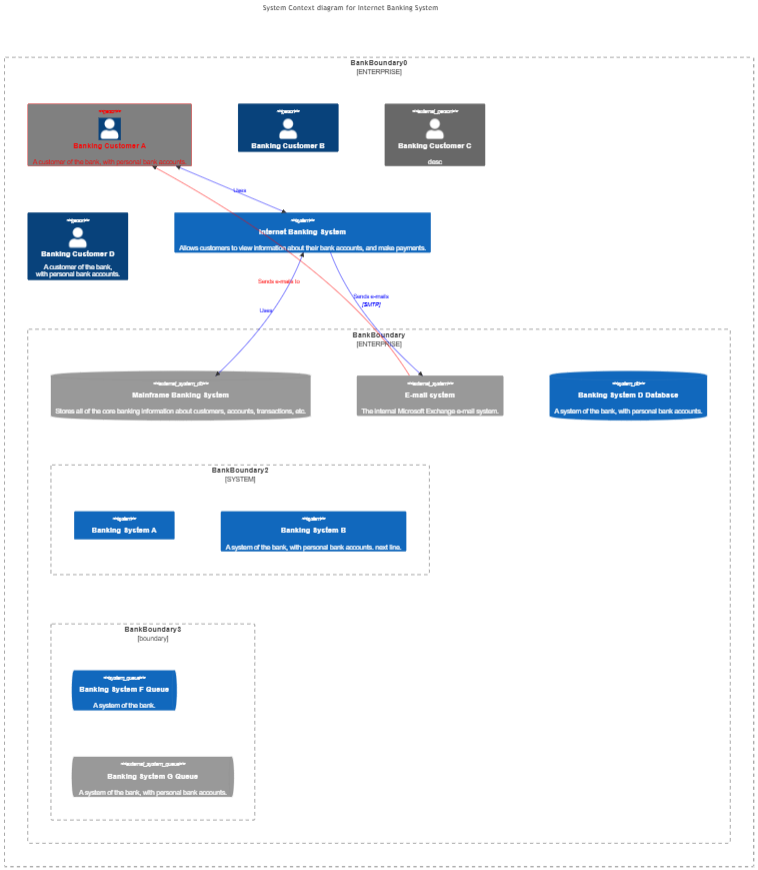
```
C4Context
  title System Context diagram for Internet Banking System
  Enterprise_Boundary(b0, "BankBoundary0") {
    Person(customerA, "Banking Customer A", "A customer of the bank, with personal ba
    Person(customerB, "Banking Customer B")
    Person_Ext(customerC, "Banking Customer C", "desc")

    Person(customerD, "Banking Customer D", "A customer of the bank, <br/> with perso

    System(SystemAA, "Internet Banking System", "Allows customers to view information

    Enterprise_Boundary(b1, "BankBoundary") {

      SystemDb_Ext(SystemE, "Mainframe Banking System", "Stores all of the core banki

      System_Boundary(b2, "BankBoundary2") {
        System(SystemA, "Banking System A")
        System(SystemB, "Banking System B", "A system of the bank, with personal bank
      }

      System_Ext(SystemC, "E-mail system", "The internal Microsoft Exchange e-mail sy
      SystemDb(SystemD, "Banking System D Database", "A system of the bank, with perso

      Boundary(b3, "BankBoundary3", "boundary") {
        SystemQueue(SystemF, "Banking System F Queue", "A system of the bank.")
        SystemQueue_Ext(SystemG, "Banking System G Queue", "A system of the bank, with
      }
    }
  }

  BiRel(customerA, SystemAA, "Uses")
  BiRel(SystemAA, SystemE, "Uses")
  Rel(SystemAA, SystemC, "Sends e-mails", "SMTP")
  Rel(SystemC, customerA, "Sends e-mails to")

  UpdateElementStyle(customerA, $fontColor="red", $bgColor="grey", $borderColor="red"
  UpdateRelStyle(customerA, SystemAA, $textColor="blue", $lineColor="blue", $offsetX=
  UpdateRelStyle(SystemAA, SystemE, $textColor="blue", $lineColor="blue", $offsetY="-
  UpdateRelStyle(SystemAA, SystemC, $textColor="blue", $lineColor="blue", $offsetY="-
  UpdateRelStyle(SystemC, customerA, $textColor="red", $lineColor="red", $offsetX="-5

  UpdateLayoutConfig($c4ShapeInRow="3", $c4BoundaryInRow="1")
```

Sample Diagrams

History

Actions

Diagram | Rough | Pan & Zoom | FULL SCREEN | SAVE TO MERMAID CHART



Structurizr DSL documentation

Upload | Render

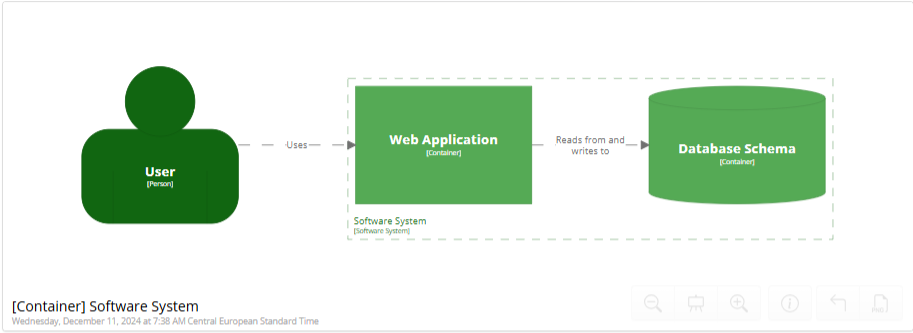
```
workspace "Name" "Description"

    !identifiers hierarchical

    model {
        u = person "User"
        ss = softwareSystem "Software System" {
            wa = container "Web Application"
            db = container "Database Schema" {
                tags "Database"
            }
        }

        u -> ss.wa "Uses"
        ss.wa -> ss.db "Reads from and writes to"
    }

    views {
        systemContext ss "Diagram1" {
            include *
            autolayout lr
        }

        container ss "Diagram2" {
            include *
            autolayout lr
        }

        styles {
            element "Element" {
                color white
            }
            element "Person" {
                background #116611
                shape person
            }
            element "Software System" {
                background #2D882D
            }
            element "Container" {
                background #55aa55
            }
            element "Database" {
                shape cylinder
            }
        }
    }

}
```

Structurizr DSL v3.2.0 - some features (e.g. `!docs`, `!adrs`, `!script`, etc) are unavailable via this demo page; see DSL for details.

Examples: Big Bank plc | Microservices | Amazon Web Services

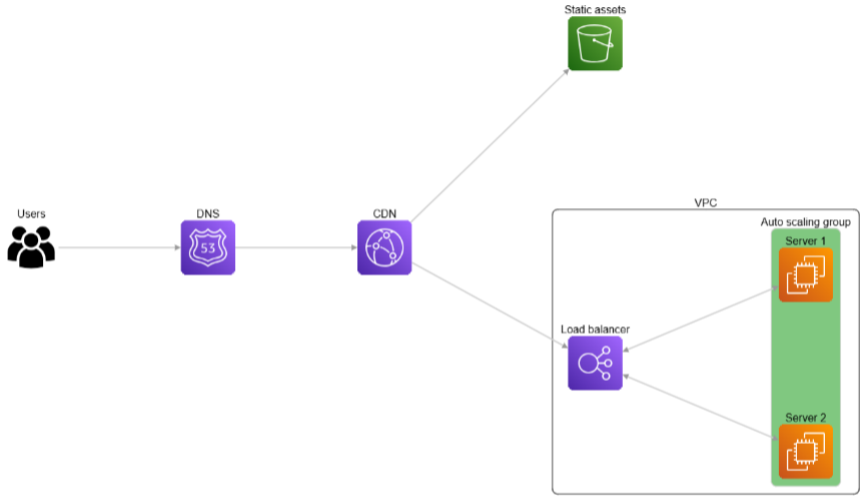
Structurizr Diagram | Structurizr Graph | Export PlantUML | Export C4-PlantUML | Export Mermaid | Export DOT | Export WebSequenceDiagrams | Export Ilograph

[Container] Software System (#Diagram2)

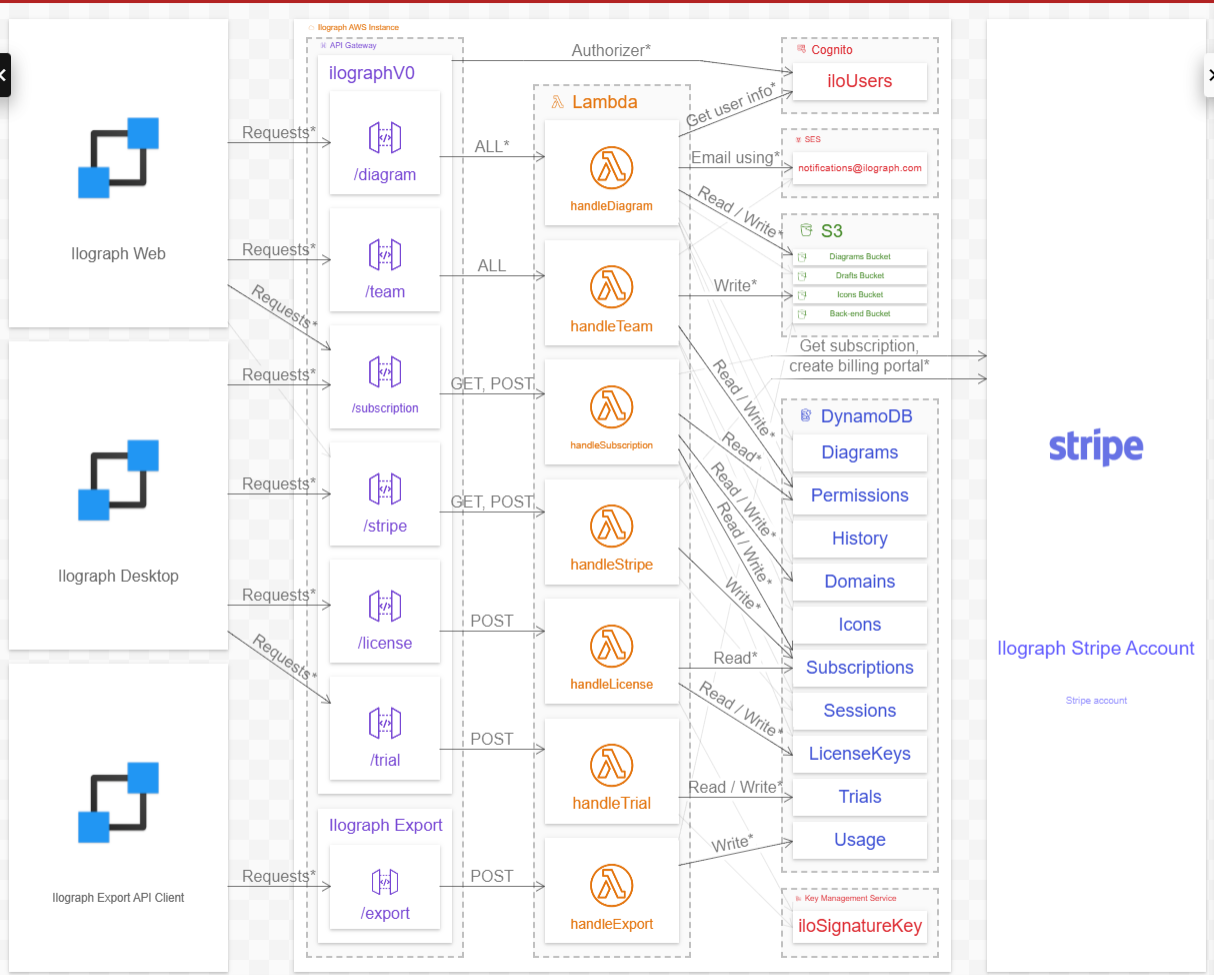


Embed | Copy link | png | Save

```
diagram "example" [direction=lr] {
  // creating the nodes
  generic.users Users;
  aws.route53 DNS;
  aws.cloudfront cf [label="CDN"];
  aws.s3 s3 [label="Static assets"];

  group vpc [label="VPC",style=solid,stroke=black,opacity=0] {
      aws.elb load_balancer [label="Load balancer"];

      group asg [label="Auto scaling group",style=solid,fill="#80c880"] {
          aws.ec2 server1 [label="Server 1"];
          aws.ec2 server2 [label="Server 2"];
      }
  }

  // creating the edges
  Users -> DNS -> cf -> load_balancer <=> asg;
  cf -> s3;
}
```



```
            *confirmUser* Lambda function. This
            function is passed the confirmed
            user's email address and username.

          This function performs three
            bookkeeping tasks for new users.

      - highlight: Permissions
        text: |-
          First, *confirmUser* checks if there
            are any pending permissions that
            need to be created for the new user
            . This occurs if the new user has
            been invited to view existing
            diagram(s) and/or join an existing
            team.

          If any are found, the appropriate
            permission(s) are written to the
            *Permissions* table.

      - highlight: notifications@ilograph.com
        text: |-
          Next, a welcome email is sent to the
            new user. This email contains links
            to Ilograph documentation and
            sample diagrams.

      - highlight: Ilograph Users
        text: |-
          Lastly, if the user has signed up for
            Ilograph updates, their email
            address will be added to the
            *Ilograph Users* contact list on
            MailChimp.

      - select: iloUsers
        highlight: Pre Token Generation
        text: |-
          Next, we'll look at the log-in
            trigger.

      - select: generateToken
        text: |-
          *Pre Token Generation* calls the
            *generateToken* Lambda function.
            This function is passed the logging
            -in user's username.

          This function generates the user's
            permissions by querying the
            *Permissions* and *Domains* tables.

          Those permissions are converted to
            claims, signed by Cognito, and
            returned to the user for.

description: |-

  This Ilograph diagram documents the
    Ilograph Serverless back-end from eleven
    perspectives. Select a perspective below
    to get started.

  *©2022 Ilograph LLC. Stripe, Namecheap,
    Mailchimp, AWS, and their icons are
    trademarks of their respective trademark
    holders.*
```



Start Walkthrough

Perspective notes

This perspective shows the run-time dependencies between resources when fulfilling requests to the Ilograph back-end.

The back-end uses a classic AWS Serverless architecture, with an *API Gateway* API calling *Lambda* functions backed by *DynamoDB* tables and *S3* buckets.

Requests related to purchasing and licenses also have an external dependency on Stripe.

DSL to Python trend

Infrastructure as Code (IaC):
Terraform --> Pulumi, AWS Cloud Development Kit (CDK)

Workflow as Code:
Oozie, Azkaban, Mistral --> Airflow, Temporal

Architecture as Code (AaC):
Draw.io (diagrams.net) --> Diagrams (Python)

02

Problem statement

Do you use UML?

Do you use **MS Visio**?

Do you use **Draw.io**?

Do you use **whiteboard**?

UML, **PowerPoint**, **Visio**, and even **Draw.io** are not good enough for Engineers to create system architecture in a fun way.

03

Solution

Architecture as Code (AaC) is easy to create, diff, version control, collaborate on, integrate into CI/CD pipeline.

Architecture as Code

pros:

- track architecture diagram changes with version control
- cloud agnostic (AWS, Azure, GCP, etc.)
- reuse of source code and collaboration
- better for tax deduction
- expect coding 🚀

cons:

- better for up-front design than long-lived documentation
- for Engineers
- whiteboard is better for prototyping

04

Diagrams Python lib

diagrams

Public

Watch 406 Fork 2.6k Starred 39.9k

master 8 Branches 44 Tags Go to file t Add file Code

tvqphuoc01 and dependabot[bot] Merge pull request #1078 from mingram... 164a2d0 · 40 minutes ago 590 Commits

.github	add pre-commit to verify commit before create PR (#1066)	2 weeks ago
assets/img	fix: resize logo	4 years ago
diagrams	Add DynamoDB Streams icon (#1074)	2 weeks ago
docker/dev	chore(deps): bump python in /docker/dev (#1079)	yesterday
docs	Add DynamoDB Streams icon (#1074)	2 weeks ago
resources	Add DynamoDB Streams icon (#1074)	2 weeks ago
scripts	add pre-commit to verify commit before create PR (#1066)	2 weeks ago
templates	Improve wording of documentation (#990)	8 months ago
tests	add pre-commit to verify commit before create PR (#1066)	2 weeks ago
website	Add DynamoDB Streams icon (#1074)	2 weeks ago
.gitignore	Add programming languages and frameworks (#112)	4 years ago
.isort.cfg	add pre-commit to verify commit before create PR (#1066)	2 weeks ago
.pre-commit-config.yaml	add pre-commit to verify commit before create PR (#1066)	2 weeks ago
CHANGELOG.md	add pre-commit to verify commit before create PR (#1066)	2 weeks ago
CONTRIBUTING.md	Improve wording of documentation (#990)	8 months ago
DEVELOPMENT.md	add pre-commit to verify commit before create PR (#1066)	2 weeks ago

About

Diagram as Code for prototyping cloud system architectures

diagrams.mingrammer.com

graphviz diagram architecture diagram-as-code

Readme
MIT license
Activity
39.9k stars
406 watching
2.6k forks
Report repository

Releases 43

v0.24.2 Latest
2 weeks ago

+ 42 releases

Used by 1.7k

+ 1,669

Contributors 152

Diagrams Python lib

```python
from diagrams import Diagram
from diagrams.aws.compute import EC2
from diagrams.aws.database import RDS
from diagrams.aws.network import ELB

with Diagram("Example 1", direction="TB", show=False):
    ELB("lb") >> [EC2("worker1"),
                  EC2("worker2"),
                  EC2("worker3"),
                  EC2("worker4"),
                  EC2("worker5")] >> RDS("events")
```

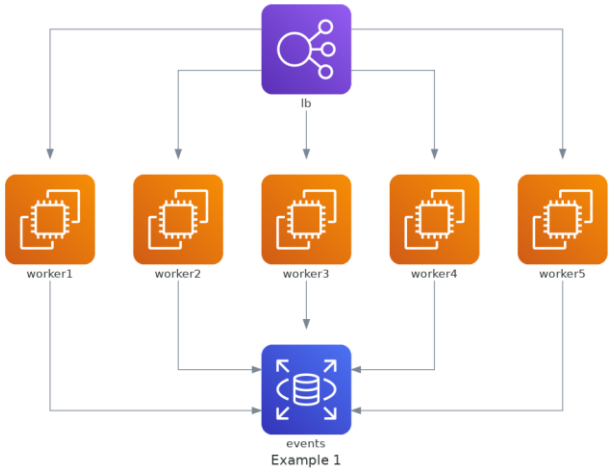


Diagram class

```
from diagrams import Diagram
from diagrams.aws.compute import EC2

graph_attr = {
    "bgcolor": "white"
}

with Diagram(
        name="Example 2",
        filename="example2",
        outformat=["png", "dot"],
        direction="LR",
        graph_attr=graph_attr,
        show=False):
    EC2("web")
```

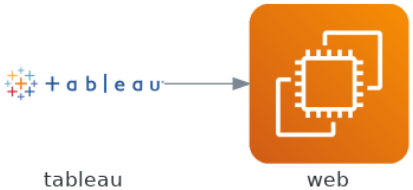
Example 2

Nodes

```python
from diagrams import Diagram
from diagrams.aws.compute import EC2

with Diagram("Example 3", filename="example3", show=False):
    EC2("web")
```

Custom nodes

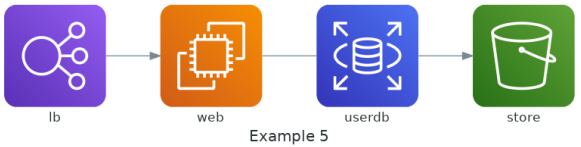
```python
from diagrams import Diagram
from diagrams.aws.compute import EC2
from diagrams.custom import Custom

with Diagram("Example 4", filename="example4", show=False):
    tableau = Custom("tableau", "tableau_logo.png")

    tableau >> EC2("web")
```



Example 4

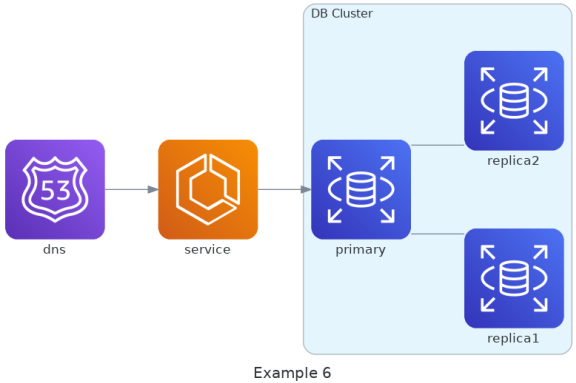
Data flow

```
from diagrams import Diagram
from diagrams.aws.compute import EC2
from diagrams.aws.database import RDS
from diagrams.aws.network import ELB
from diagrams.aws.storage import S3

with Diagram("Example 5", filename="example5", show=False)
    elb = ELB("lb")
    ec2 = EC2("web")
    rds = RDS("userdb")
    s3 = S3("store")

    (elb >> ec2) - rds >> s3
```



Clusters

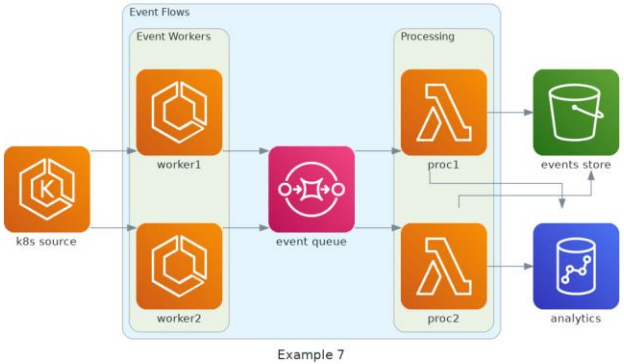
```
from diagrams import Cluster, Diagram
from diagrams.aws.compute import ECS
from diagrams.aws.database import RDS
from diagrams.aws.network import Route53

with Diagram("Example 6", filename="example6", show=False):
    dns = Route53("dns")
    web = ECS("service")

    with Cluster("DB Cluster"):
        db_primary = RDS("primary")
        db_primary - [RDS("replica1"),
                      RDS("replica2")]

    dns >> web >> db_primary
```



Example 6

Nested clusters

```
from diagrams import Cluster, Diagram
from diagrams.aws.compute import ECS, EKS, Lambda
from diagrams.aws.database import Redshift
from diagrams.aws.integration import SQS
from diagrams.aws.storage import S3

with Diagram("Example 7", filename="example7", show=False):
    source = EKS("k8s source")

    with Cluster("Event Flows"):
        with Cluster("Event Workers"):
            workers = [ECS("worker1"),
                       ECS("worker2")]

        queue = SQS("event queue")

        with Cluster("Processing"):
            handlers = [Lambda("proc1"),
                        Lambda("proc2")]

    store = S3("events store")
    dw = Redshift("analytics")

    source >> workers >> queue >> handlers
    handlers >> store
    handlers >> dw
```



Example 7

05

Closing slides

architecture-as-code-python-summit-2024

Public

Pin Unwatch 1 Fork 0 Star 0

main 1 Branch 0 Tags Go to file t Add file Code

korniichuk Initial commit 69a92be · 1 minute ago 2 Commits

.gitignore	Initial commit	9 minutes ago
LICENSE	Initial commit	9 minutes ago
README.md	Initial commit	9 minutes ago
d1_example.py	Initial commit	1 minute ago
d2_diagram_class.py	Initial commit	1 minute ago
d3_nodes.py	Initial commit	1 minute ago
d4_custom_nodes.py	Initial commit	1 minute ago
d5_data_flow.py	Initial commit	1 minute ago
d6_clusters.py	Initial commit	1 minute ago
d7_nested_clusters.py	Initial commit	1 minute ago
example2.dot	Initial commit	1 minute ago

About

Architecture as Code (AaC) with Python

- Readme
- Unlicense license
- Activity
- 0 stars
- 1 watching
- 0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Languages

tinyurl.com/PYAAC

RUSLAN KORNIICHUK

Software and Platform Engineer in Poland

Hire me

Solid experience of 9+ years in IT. Former Software Engineer and Architect at Fortune 500 companies. Core areas: Cloud Computing, and Software Engineering. Especially interested in Platform Engineering. Click the button above to hire me.

DONATE TO UKRAINE ARMY FUNDRAISER

Select the destination of your assistance

ARMY ASSISTANCE | SUPPORT THE FUND

All funds raised to help the army will be spent exclusively on purchases for the Armed Forces of Ukraine and other components of the Defense Forces.

Payment method

PAYMENT BY CARD | SWIFT TRANSFERS | CRYPTOCURRENCY

Regularity

ONE TIME | SUBSCRIPTION EVEN BETTER

Payment currency

UAH, ₴ | USD, $ | EUR, €

Amount of donation*

€ 10

Email*

example@example.com

☐ I consent to the processing of my email address to receive email newsletters (twice a month), invitations and receipts.

PAY

Payment secured

06

Q&A session