Why and How We Went Serverless, and How You Can Too

CNI Spring 2021 Virtual Membership Meeting

Yinlin Chen and Bill Ingram
Virginia Tech Libraries
March 15-26, 2021

The Presenters

Yinlin Chen

Digital Library Architect And
Assistant Professor
University Libraries,
Virginia Tech

William Ingram

Assistant Dean
University Libraries,
Virginia Tech

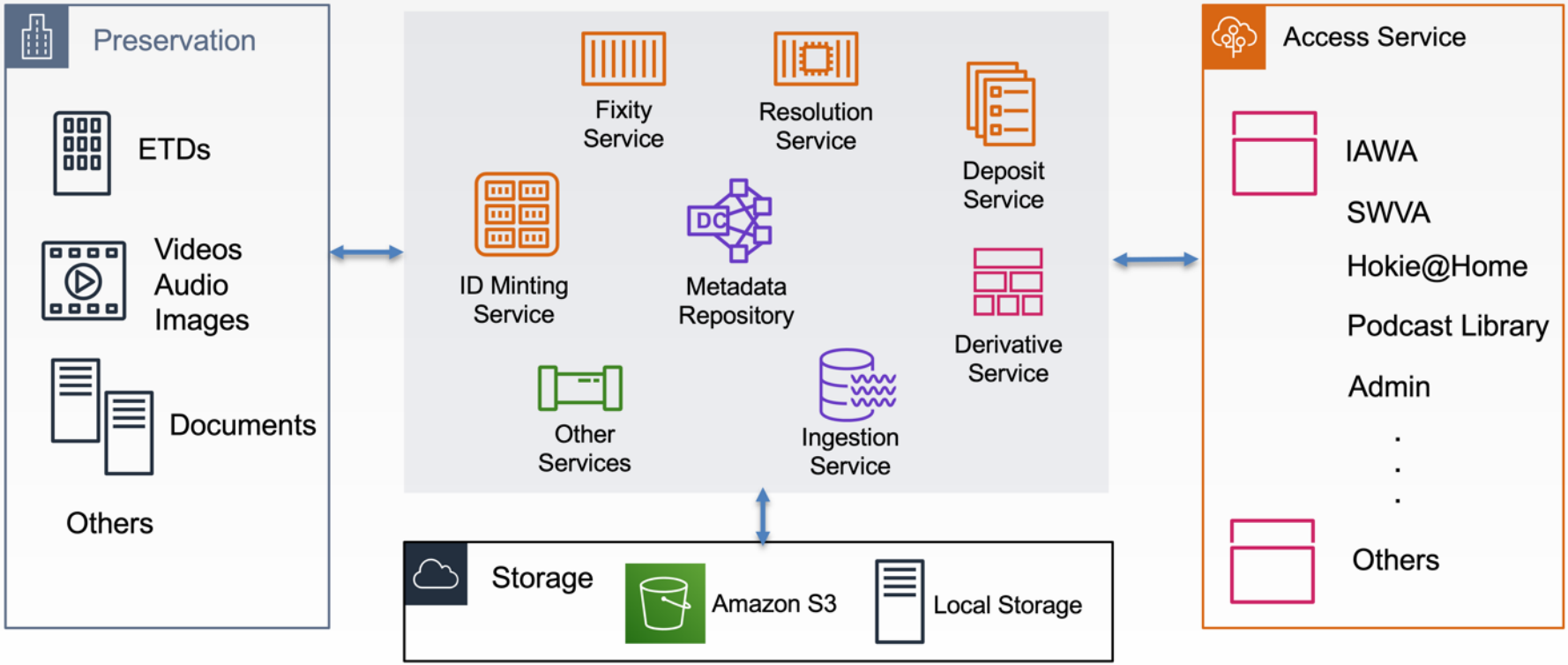
Outline

- Challenges
- Virginia Tech Digital Library Platform
- Methodology
- What we did: Before vs Now
- Conclusion
- Future work

Challenges

- Numerous, web applications with similar stacks stretching resources
- Limited in-house capacity to address performance, resilience, and scaling
- Library-specific software requires training or competing for few experienced library Software developers and Sysops
- The complexity nature to modify or add new feature in the monolithic applications

Virginia Tech Digital Library Platform



Methodology

The twelve-factor app

1.Codebase
One codebase, many deploys

2. Dependencies
Explicitly declare and isolate dependencies

3. Config
Store config in the environment

4. Backing services
Treat backing services as attached resources

5. Build, release, run
Strictly separate build and run stages

6. Processes
Execute the app as one or more stateless processes

7. Port binding
Export services via port binding

8. Concurrency
Scale out via the process model

9. Disposability
Maximize robustness with fast startup and graceful shutdown

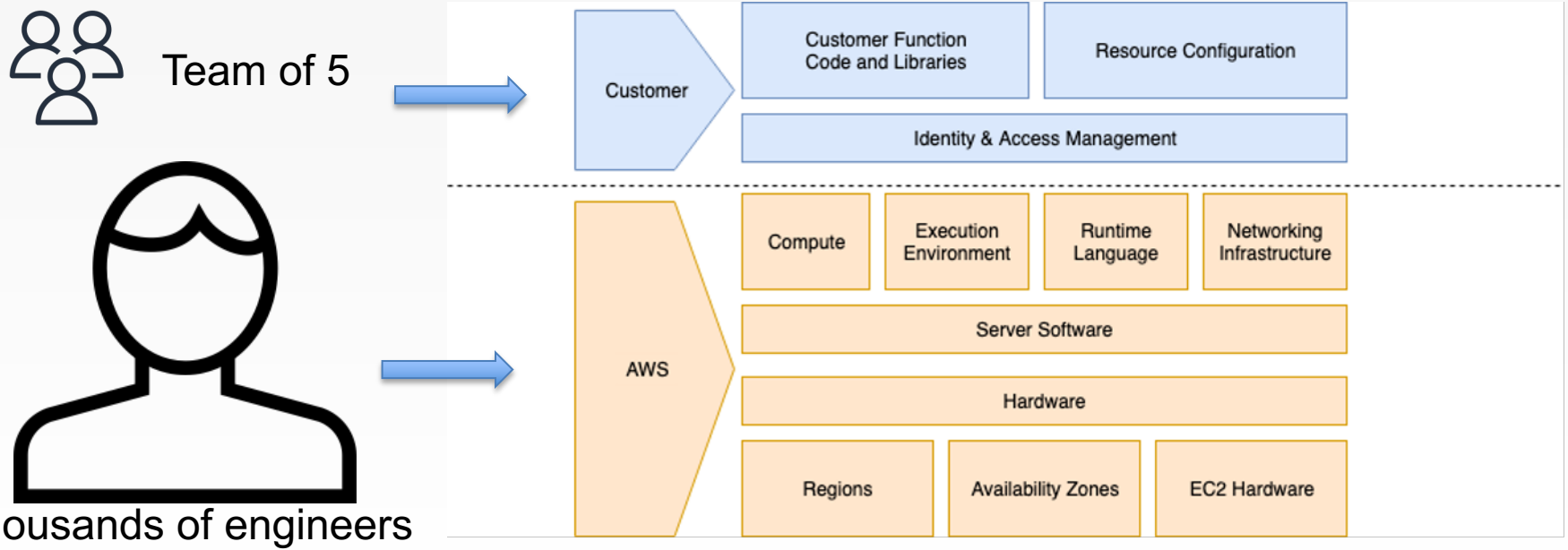
10. Dev/prod parity
Keep development, staging, and production as similar as possible

11. Logs
Treat logs as event streams

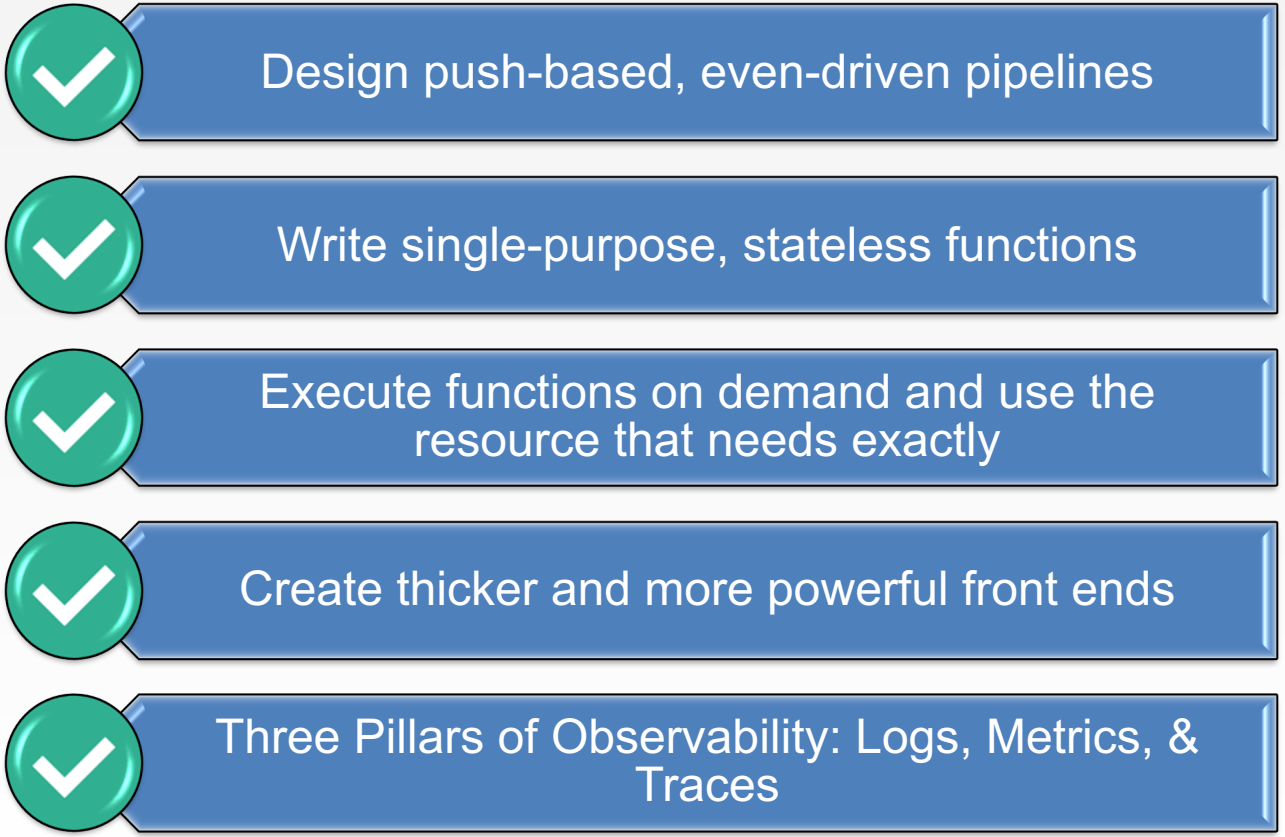
12. Admin processes
Run admin/management tasks as one-off processes

If I have seen further, it is by
standing on the shoulders of Giants

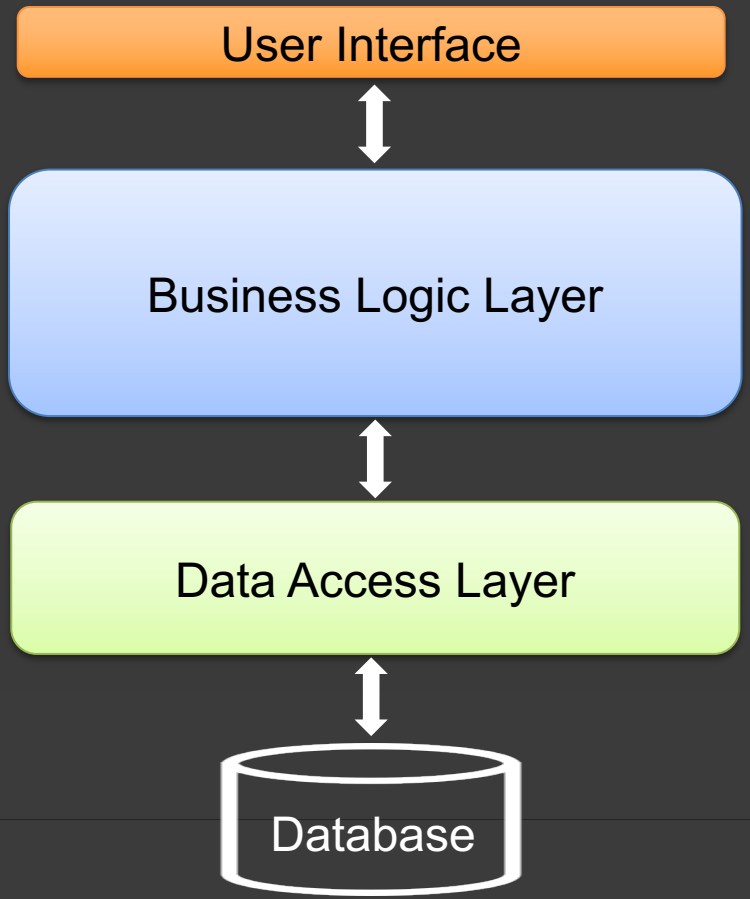
- Isaac Newton



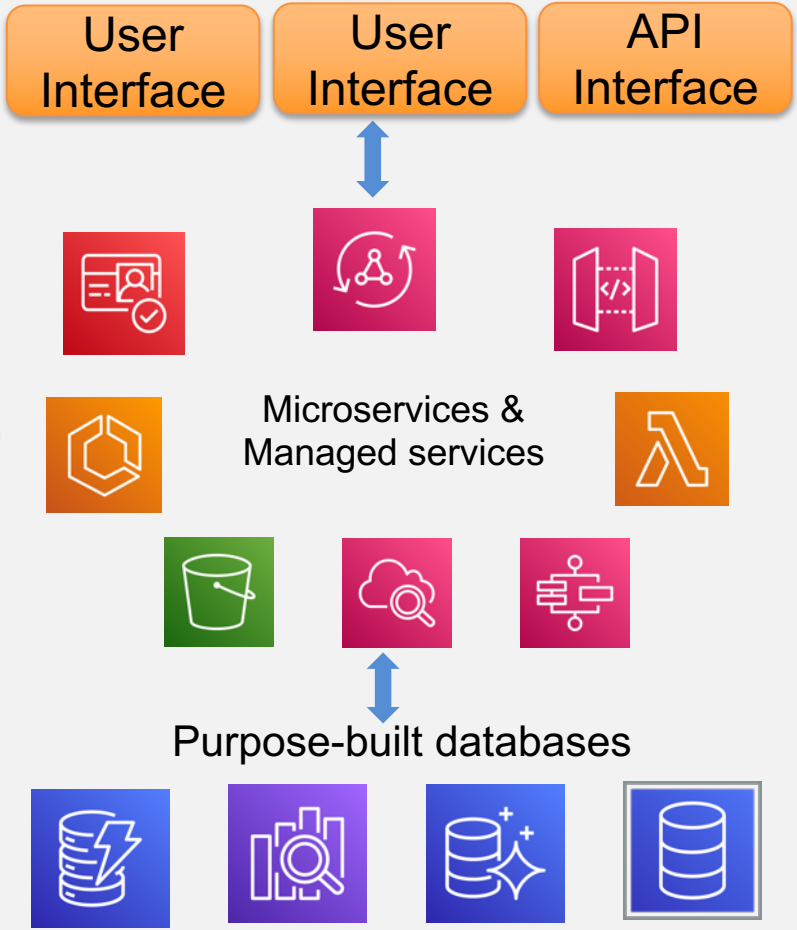
Principles of Serverless Design



Monolithic Architecture

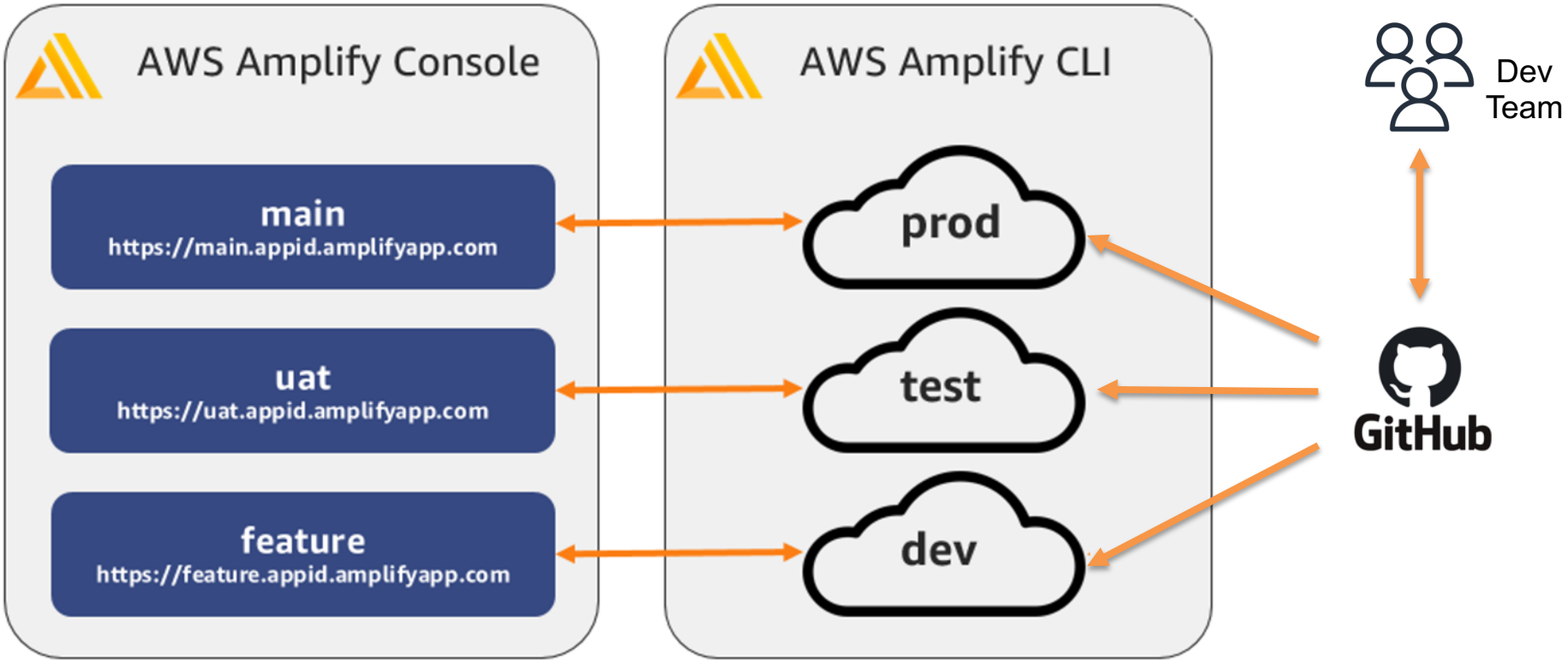


Serverless Architecture

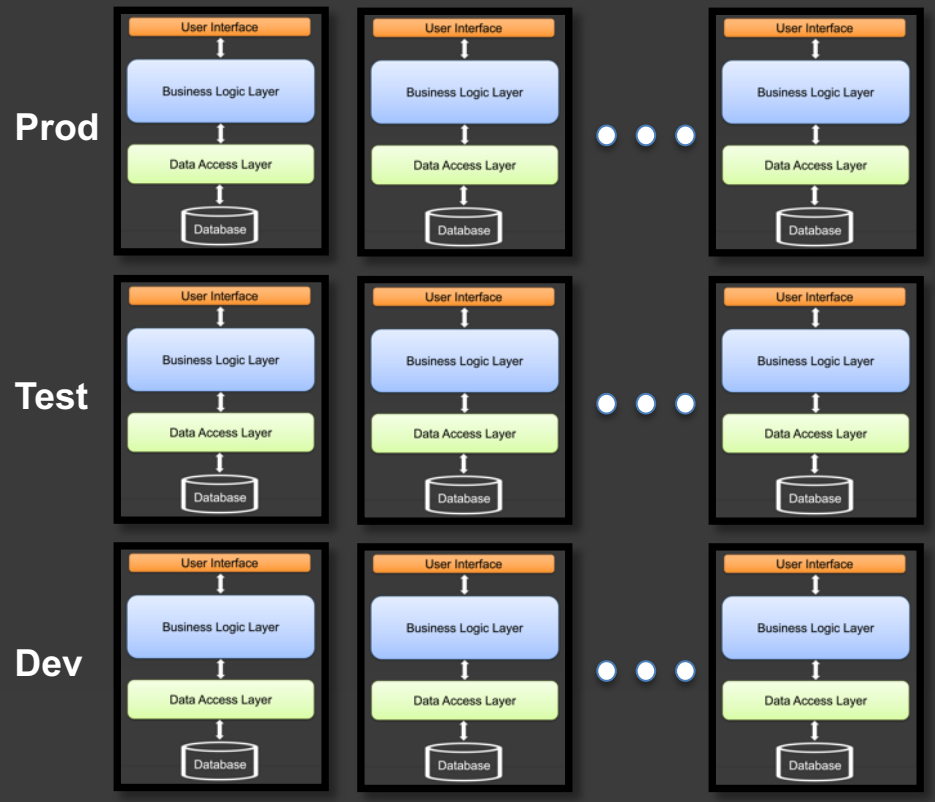


Multi-Tenant Web Applications

Multi-stage deployments through Dev/Test/Prod Environments

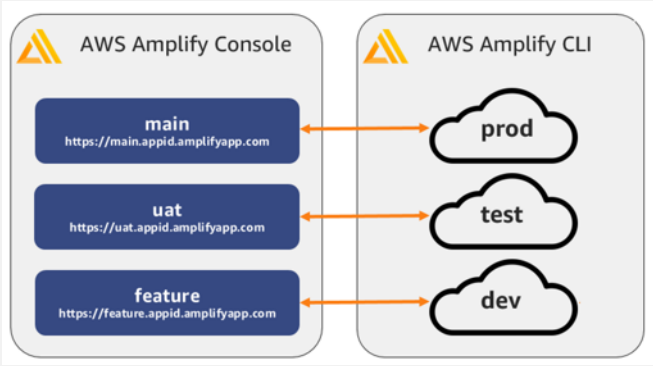


Multi-site deployment : Before



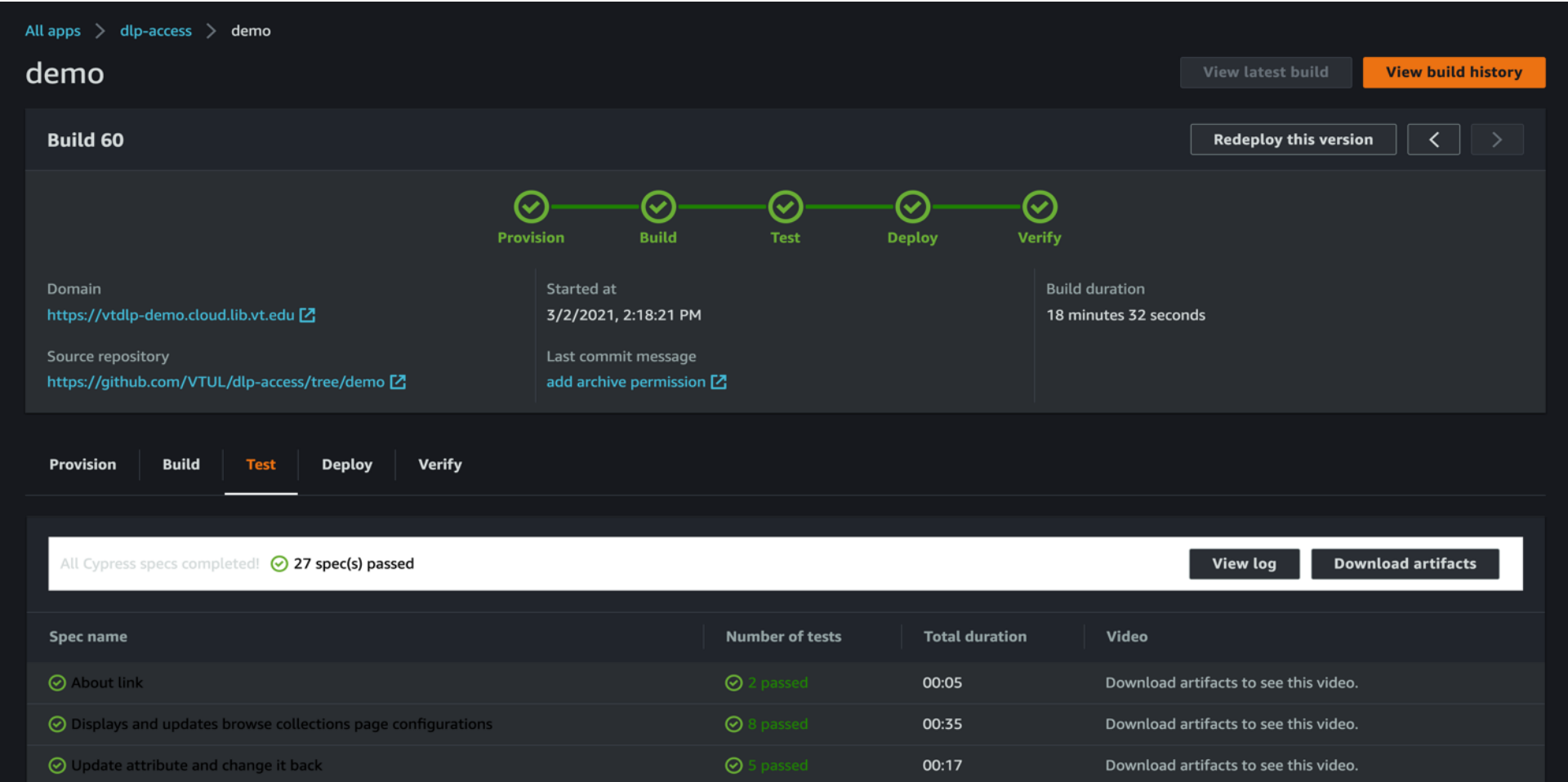
Ansible Playbooks, Shell scripts, etc.

Multi-site deployment: Now

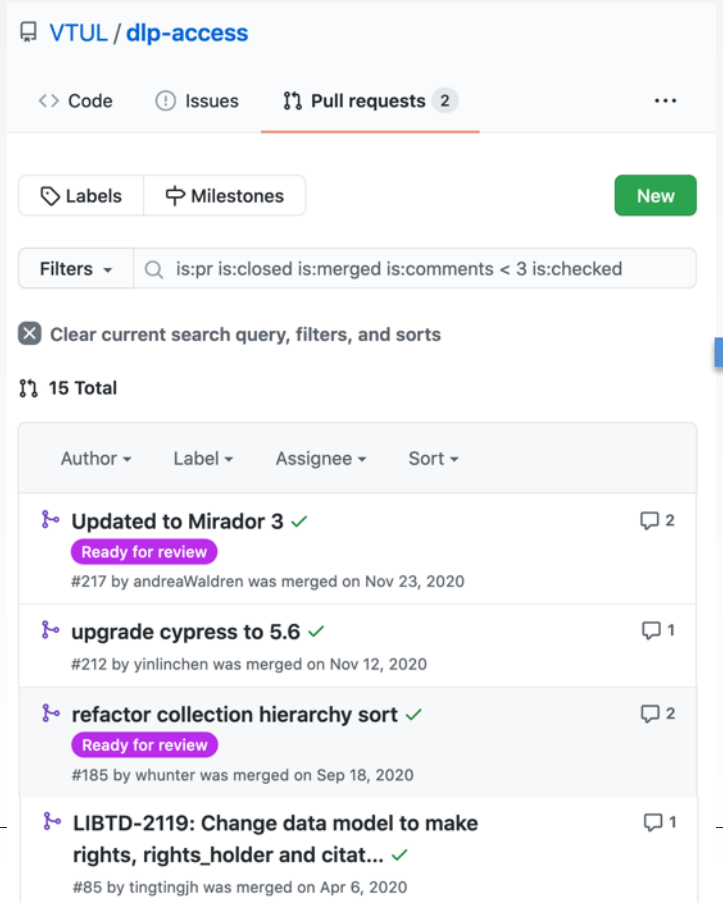


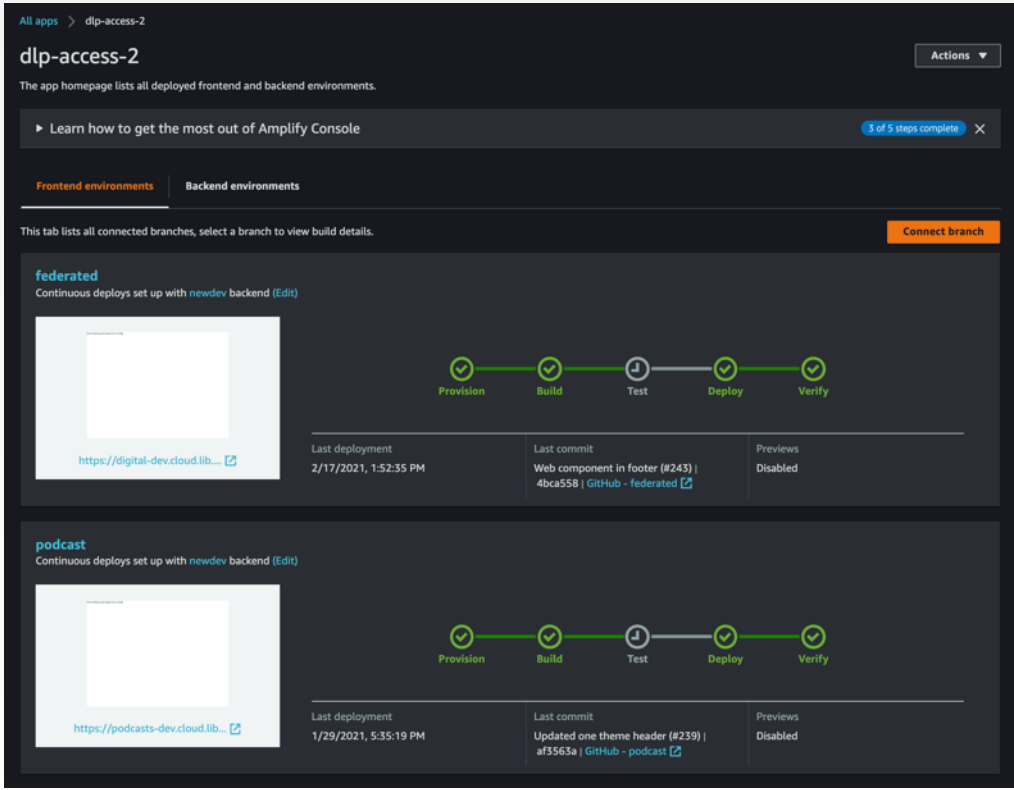
- No Servers need us to perform maintenance
- Unlimited ephemeral Test & Dev deployments
- Step up deployment and let AWS do the rest
- Initially a small instance is created and let AWS to scale it up and down for us
- … and many many more

Fully Automated CI/CD Pipeline



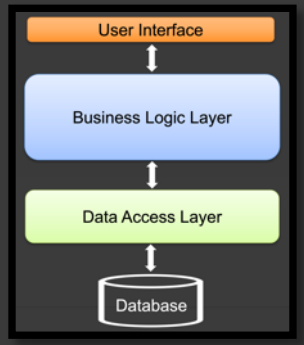
Every Single Feature (Change) has a Preview



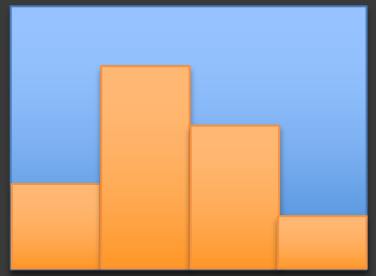


VIRGINIA TECH

Before

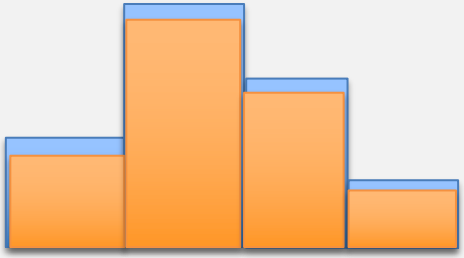


Resource Usage



Now

Resource Usage



Website performance

Page	IAWAv1	IAWAv2	Percent Change
Index Page V1 V2	796 ms	766 ms	4% faster
Item Browse Page V1 V2	1.35 s	882 ms	35% faster
Collection Browse Page V1 V2	926 ms	754 ms	20% faster
Item Page V1 V2	1.01 s	861 ms	16% faster

There are other things that we can not compare the performance, because we can only do those in AWS

Image Processing: Before

Batch Scripting Jobs in Server

We have 11 collections of images need to be processed

We only have 1 spare server for this, we will do 1 collection a time

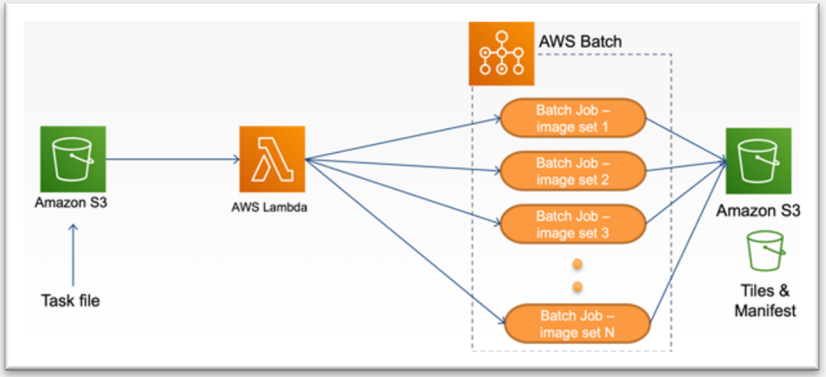
It looks like we need 7 days to process one set of images, is it OK?

That take too long, can you provision servers in AWS and do all at once?

Run 11 servers for a week for this… seems expensive and hard to manage them.

Image Processing: Now

Serverless Batch Jobs



- Processed 11 collections of images in 2 days
- Expect to process 20, 30, … collections of images will still in 2 days
- There are still space to optimize microservices and make it even faster

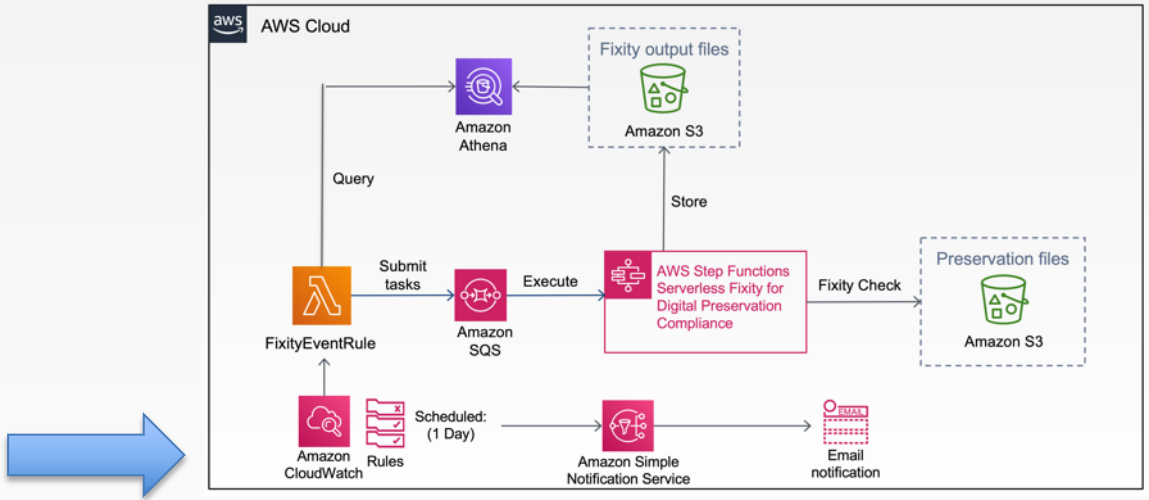
Infrastructure as Code (DevOps)

```yaml
SubmitSteps:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: fixitycheck/
    Handler: steps.lambda_handler
    Runtime: python3.8
    Timeout: 30
    Layers:
      - !Ref SharedUtils
    Policies:
      - StepFunctionsExecutionPolicy:
          StateMachineName: !Ref StateMachineName
    Events:
      FixitySQSEvent:
        Type: SQS
        Properties:
          Queue: !GetAtt FixitySqsQueue.Arn
          BatchSize: 10

    Environment:
        Variables:
          Region: !Ref Region
          StateMachineArn: !Sub "arn:aws:states:${Region}

FixitySqsQueue:
  Type: AWS::SQS::Queue

FixitySNS:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: fixitycheck/
    Handler: sns.lambda_handler
    Runtime: python3.8
    MemorySize: 2048
    Timeout: 120
    Layers:
```



Launch Stack

One click deploy

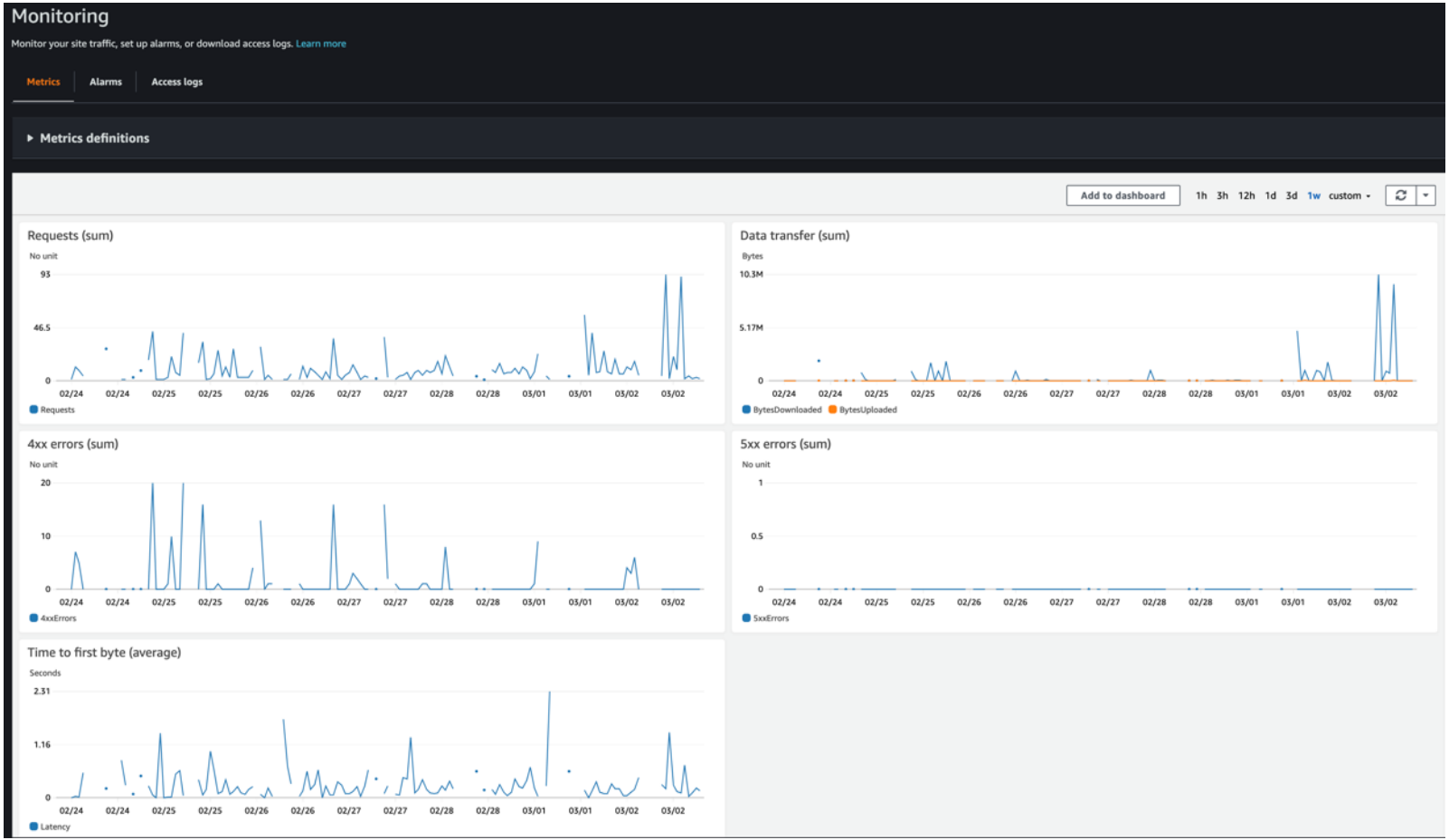
```
CloudFormation stack changeset
-------------------------------------------------------------------------------------------------------------------------------------------------------------
Operation                                                  LogicalResourceId                                          ResourceType
-------------------------------------------------------------------------------------------------------------------------------------------------------------
* Modify                                                   ConvertFileFunction                                        AWS::Lambda::Function
* Modify                                                   UploadBucket                                               AWS::S3::Bucket
-------------------------------------------------------------------------------------------------------------------------------------------------------------

Changeset created successfully. arn:aws:cloudformation:us-east-1:909117335741:changeSet/samcli-deploy1614752401/f3165ad7-122e-4644-bd52-f4b4c8b1c8b1

2021-03-03 01:20:17 - Waiting for stack create/update to complete

CloudFormation events from changeset
-------------------------------------------------------------------------------------------------------------------------------------------------------------
ResourceStatus                                             ResourceType                                               LogicalResourceId
-------------------------------------------------------------------------------------------------------------------------------------------------------------
UPDATE_IN_PROGRESS                                         AWS::Lambda::Function                                      ConvertFileFunction
UPDATE_COMPLETE                                            AWS::Lambda::Function                                      ConvertFileFunction
^[UPDATE_COMPLETE_CLEANUP_IN_PROGRESS                        AWS::CloudFormation::Stack                                 testhello
UPDATE_COMPLETE                                            AWS::CloudFormation::Stack                                 testhello
```

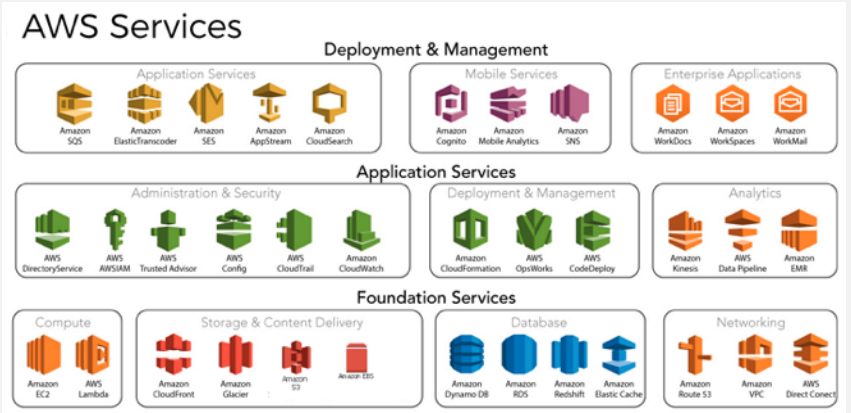
Monitor Metrics for Applications in Near Real-time



Cost: Before

- Add one server, plus the cost
- Plan server capacity each year
  - Overestimation: shortage is even worse, so would rather purchase more than we need (Wasting resource is fine, we already paid money, but really?)

Cost: Now



- Pay what we use
  - Use it well than pay less
  - Pay 0 when services are idle
- Some (invisible) cost are vanished
  - Server maintenance cost
  - Labor, time, electric, etc.

Conclusion

- AWS services just like Lego blocks, we pick and choose blocks and build powerful, scalable applications that suit our business
- With serverless, we write less code, fasten development speed, and enable creativities and innovations
- Applications we built are more resilient, secure, scalable, and cost-efficient than in a traditional, server-based environment
- AWS communities are larger than any software communities in the libraries

Image source: Lego.com

Future Work

- New features and services
  - AI/ML/DL as a service
  - Large-scale text and data mining
  - Digital preservation and the management
- Continue performance tuning and cost optimization of cloud-native services
- Continue refactoring of legacy applications to AWS
- Collaborate with other institutions on the Serverless Cloud

Thank you!

Virginia Tech Digital Libraries

Cloud-native, multi-tenancy digital library software

https://github.com/vt-digital-libraries-platform

aws-batch-iiif-generator

A project that demonstrates the use of AWS Batch to create IIIF tiles and manifests. It is used for VTDLP Derivative Service

Python

DLPServices

VTDLP Services

resolution-service

VTDLP Resolution Service

Python

dlp-access

DLP Access Website: A Multi-Tenancy Serverless web application

JavaScript

NOID-mint

Mint Nice Opaque IDentifiers

Python

FixityService

VTDLP Fixity Service: Regularly scheduled fixity checks and comparisons to the files stored in the AWS S3

JavaScript