Building Large Scale, Microservice-driven Applications

Andrei Papancea ’15
Columbia University, MS Computer Science

NLX Inc, CEO & Co-Founder

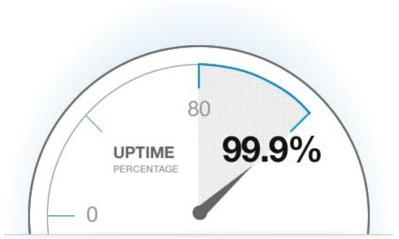
Dealing with Large Scale Applications

Channel(s)

Platform(s)

Security Requirements

Availability



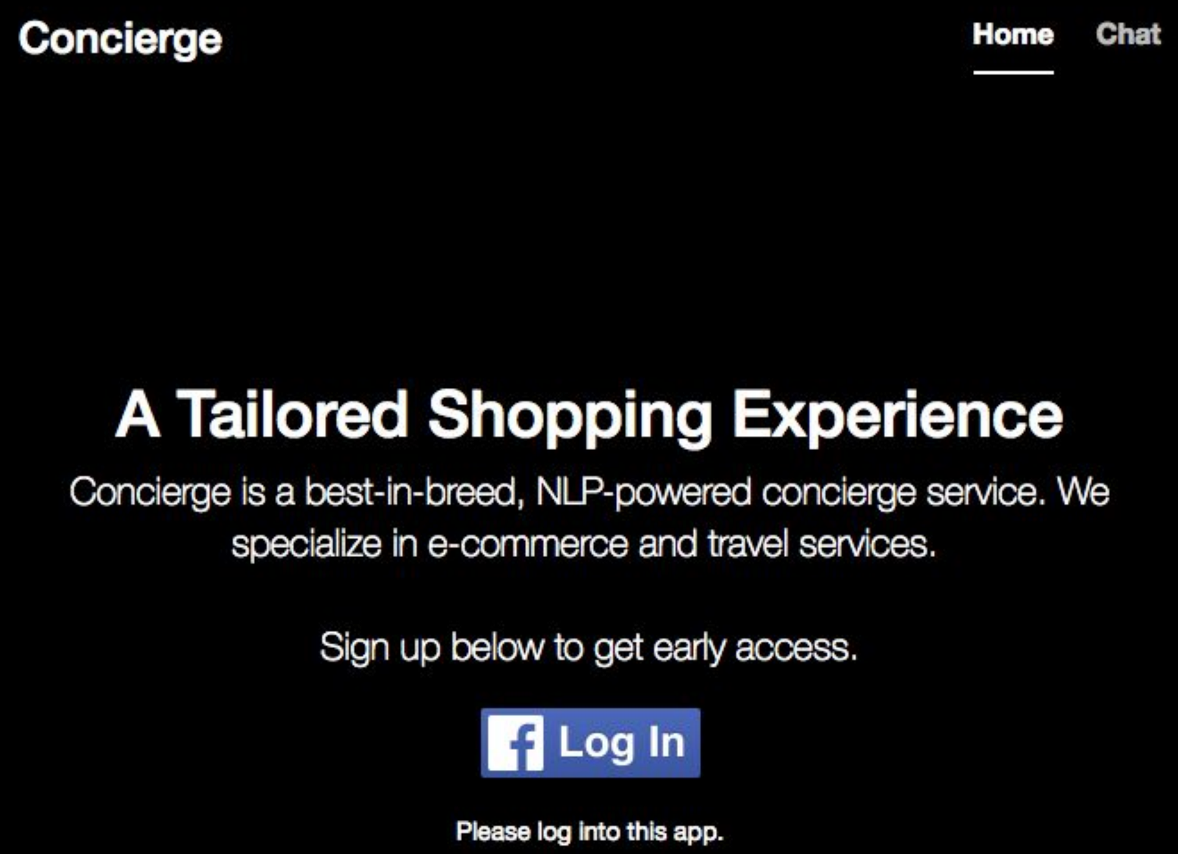
Cost

Goal:
Learn how to build highly available, distributed, and scalable systems that are also cost-effective, using Microservices.

1. Problems with Monolithic Systems
2. Microservices: A Solution
3. Microservice-driven APIs
4. Scaling the Frontend
5. Asynchronous Workflows

Scenario:
You have a great idea for a new AI-powered concierge service.

You set up a quick MVP to get your product out there.



Concierge

Home Chat

A Tailored Shopping Experience

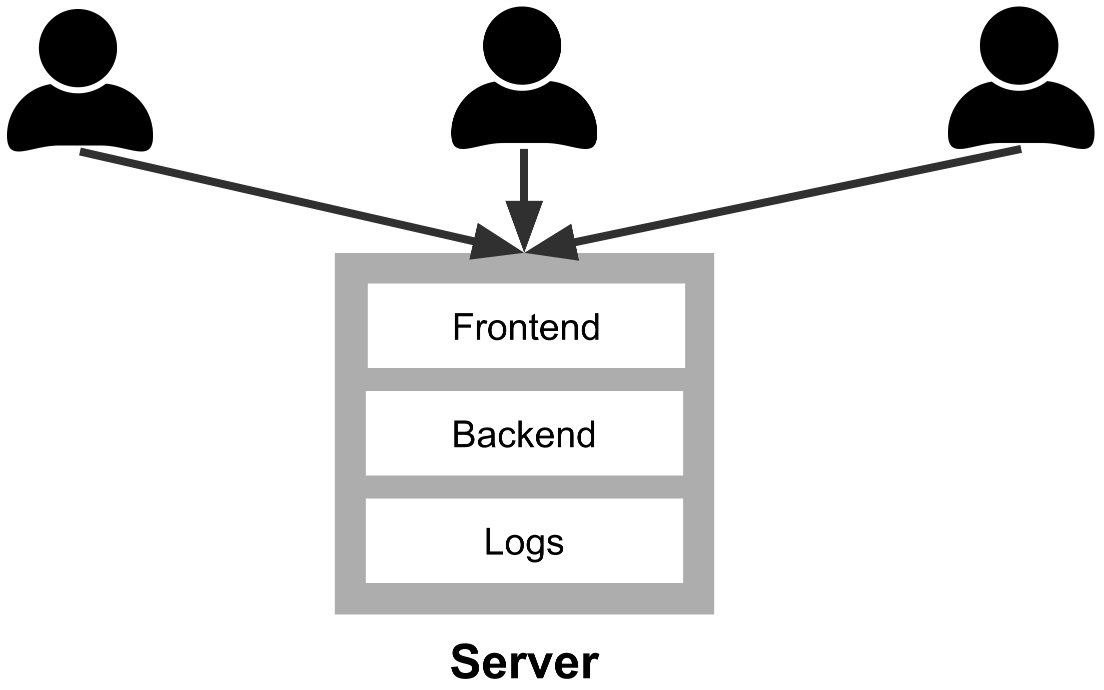
Concierge is a best-in-breed, NLP-powered concierge service. We specialize in e-commerce and travel services.

Sign up below to get early access.

Log In

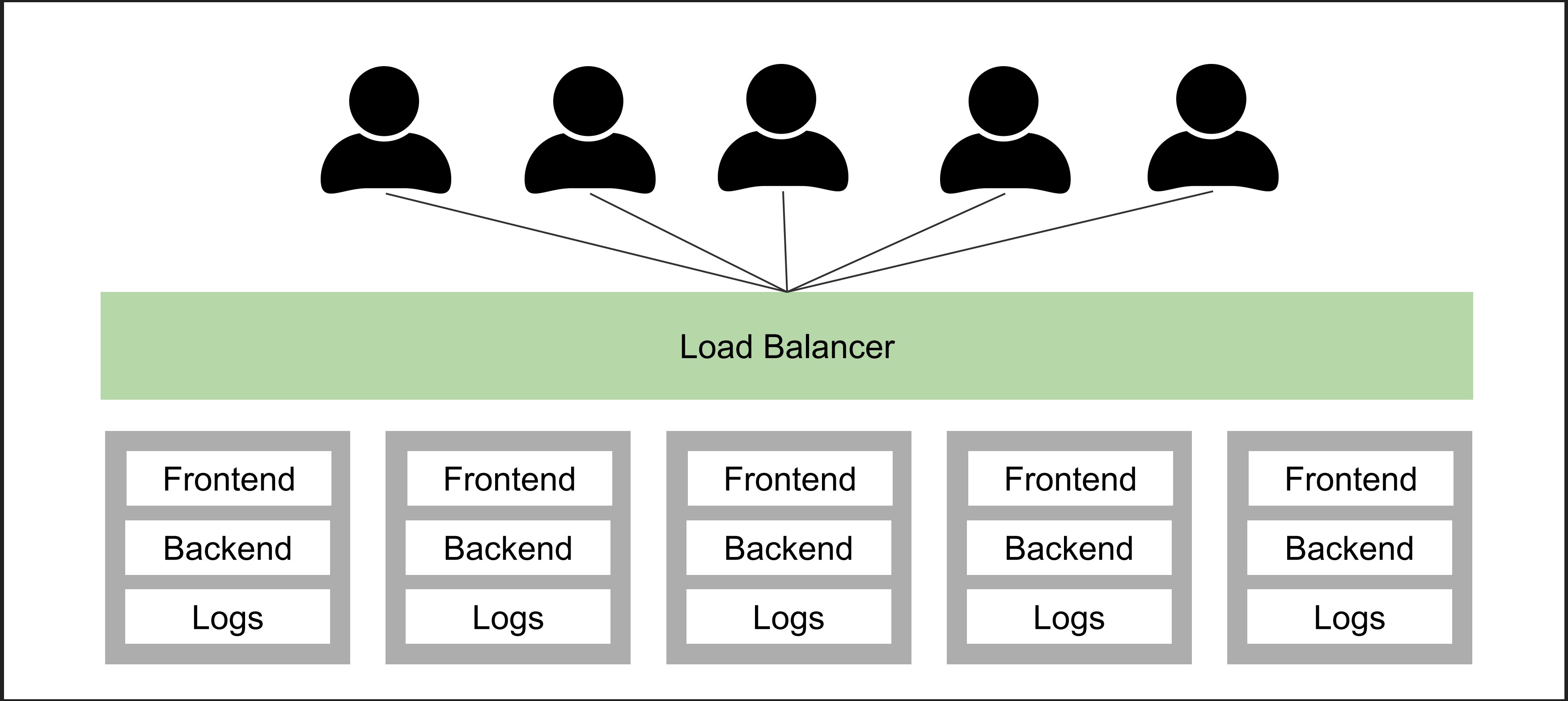
Please log into this app.

Your new website is up!



Now you scale from
100 users to <u>100k</u>.

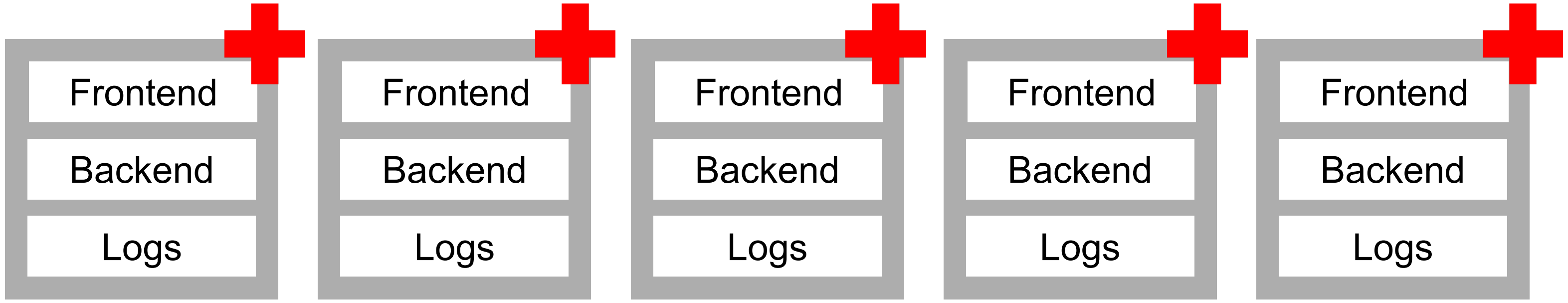
Solution? Scale out.



But, adding more servers can get very expensive.

A security backdoor
has been discovered
in the server software
that you use.

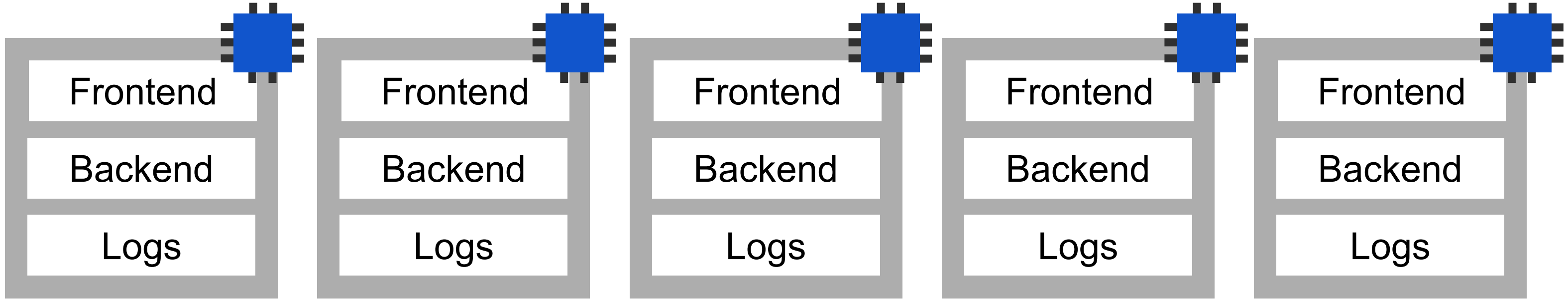
Solution? Patch all your instances.



Ugh, managing servers
is <u>time consuming</u>.

You realize your NLP operations consume too much memory.

Your solution? Increase instance memory.



Everything starts getting more and more expensive.

Alex, one of the developers, decides to deploy the feature he has been working on all week.

BUT…

...Sam committed non-working code to the master branch.

The new feature can’t be deployed until Sam fixes the bugs. <u>Frustrating</u>.

Challenges with Monolithic Systems

- Code complexity and maintainability
- **Deployment becomes the bottleneck**
- Fear to change
- **Lack of ownership**
- Failure dependencies
- **One size doesn’t fit all**
- Hard to scale out

1. Problems with Monolithic Systems
2. Microservices: A Solution
3. Microservice-driven APIs
4. Scaling the Frontend
5. Asynchronous Workflows

Microservices to the rescue!

- An architectural pattern
- Split the application into multiple services that:
    - Are <u>small</u>
    - Use <u>simple</u> protocols
    - Are <u>loosely-coupled</u>
    - Can be <u>independently deployed</u>
    - + each can be written in a different language

Benefits of Microservices

Speed

- Faster development & deployments

Innovation

- Autonomy of teams, culture of change
- Ownership and DevOps culture

Quality

- Composability and reusability
- More maintainable code
- Better scaling and optimizations
- Failure Isolation and Resiliency

Microservices++: Serverless Components

- No servers to manage
- Scalability out of the box
- Minimize codebase size
- Pay per usage
- Extremely low cost (usually fractions of a cent)

Your new best friends.

Lambda

Your new best friends.

Lambda

API Gateway

Your new best friends.

Lambda

API Gateway

Cognito

Your new best friends.

Lambda

API Gateway

Cognito

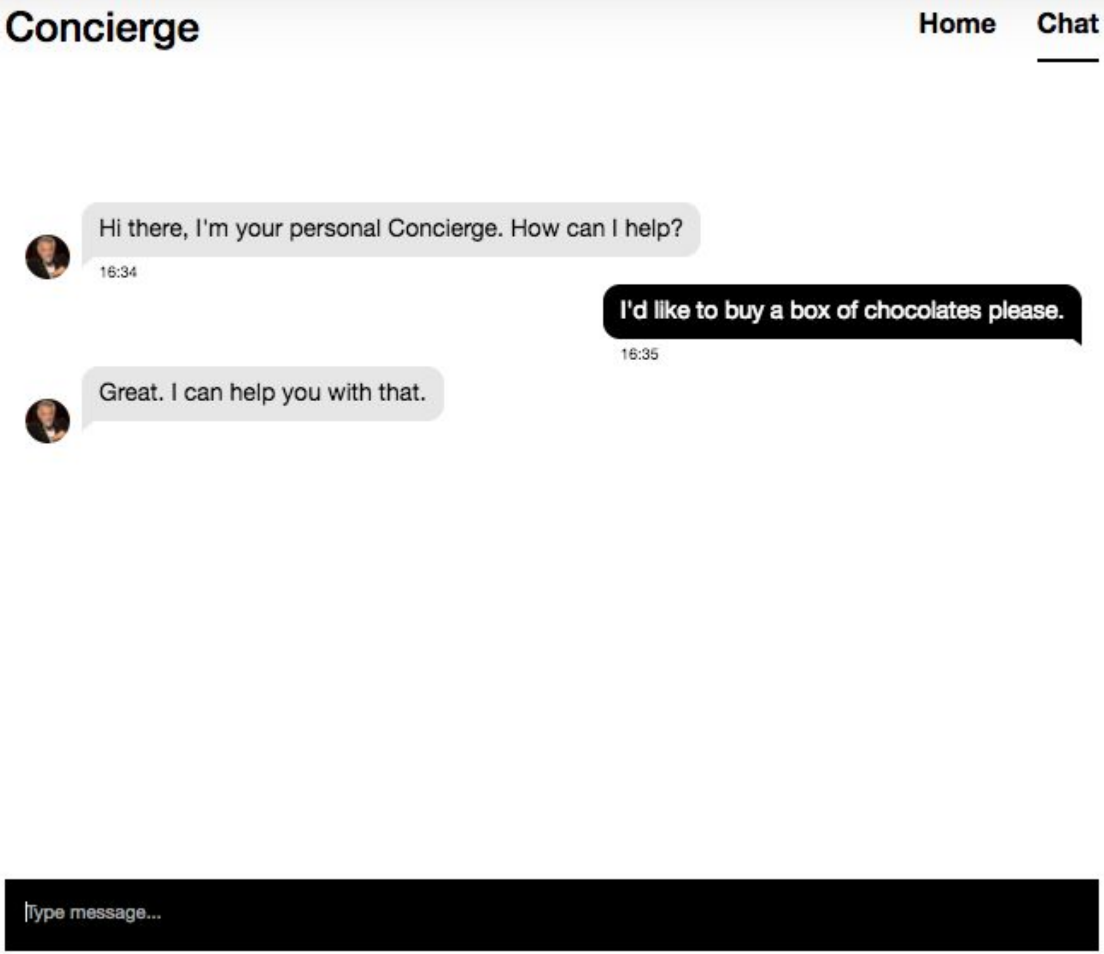
IAM

Recap:

1. Microservices are an architectural pattern used to decouple applications
2. AWS offers lots of different managed services, that can be used as building blocks in your microservice-driven systems

1. Problems with Monolithic Systems
2. Microservices: A Solution
3. Microservice-driven APIs
4. Scaling the Frontend
5. Asynchronous Workflows

Let’s build our Concierge Service.



Concierge

Home

Chat

Hi there, I'm your personal Concierge. How can I help?

16:34

I'd like to buy a box of chocolates please.

16:35

Great. I can help you with that.

Type message...

DEMO

Walkthrough of our Concierge app.

Where do I start?

Design. Design. Design.

Not (just) UI design

- Stack design
- Architecture design
- Data structure design
- API design

API Design

- forces you to think before you build
- drives a good chunk of the architecture
- drives the data structure design
- makes everything more efficient
  - no more “I’m waiting for the backend to be ready before I can start to implement the frontend”
  - minimizes time wasted restructuring the API in future versions

Yeah … but building documentation sucks and it is time consuming.

That’s why there’s Swagger.

Swagger

- “The world’s most popular API framework”
- Powerful tool to design, build, document, and consume REST APIs
- Open Source
- User friendly
- Standardized

Check out http://swagger.io

Swagger + API Gateway = ❤️

- Seamless API setup
- Import the Swagger configuration into API Gateway
  - endpoints
  - security settings
  - request/response models
  - request/response mapping
  - response codes, etc.

Swagger + API Gateway + Lambda = 🎉

- Custom Swagger definitions for Lambda
- Set up a fully integrated and managed API
- Built-in API management features
    - API keys
    - Throttling
    - Security
    - Staging, etc.

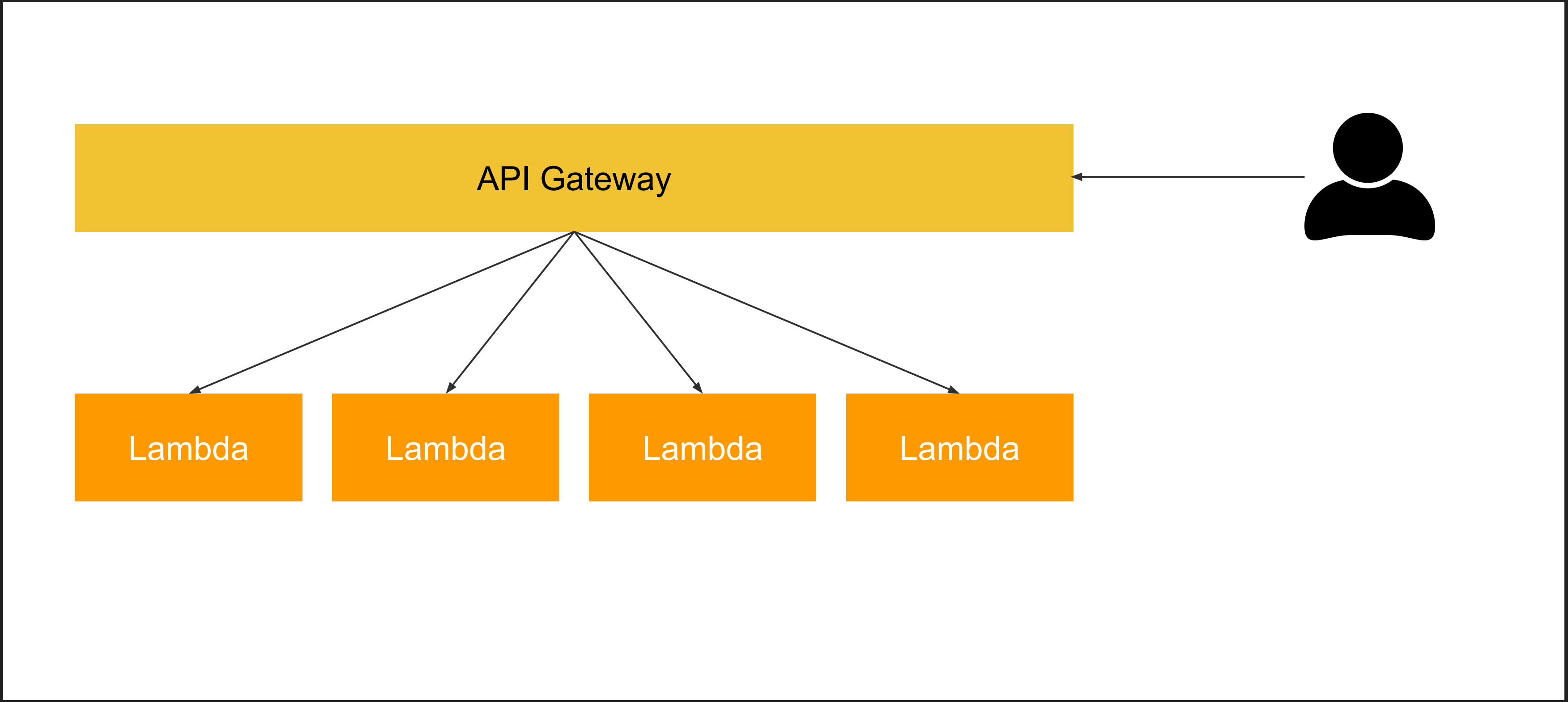
DEMO

Setup the API using Swagger, API Gateway, Lambda

Code Deployment with Lambda

- Using the AWS Web Console
- Using the AWS CLI (preferred)
- Bash script
- Check out the sample deployment script
  - https://github.com/mangatanyc/columbia-lecture-concierge/blob/master/backend/deploy.sh

Our Stack so far



What’s missing?

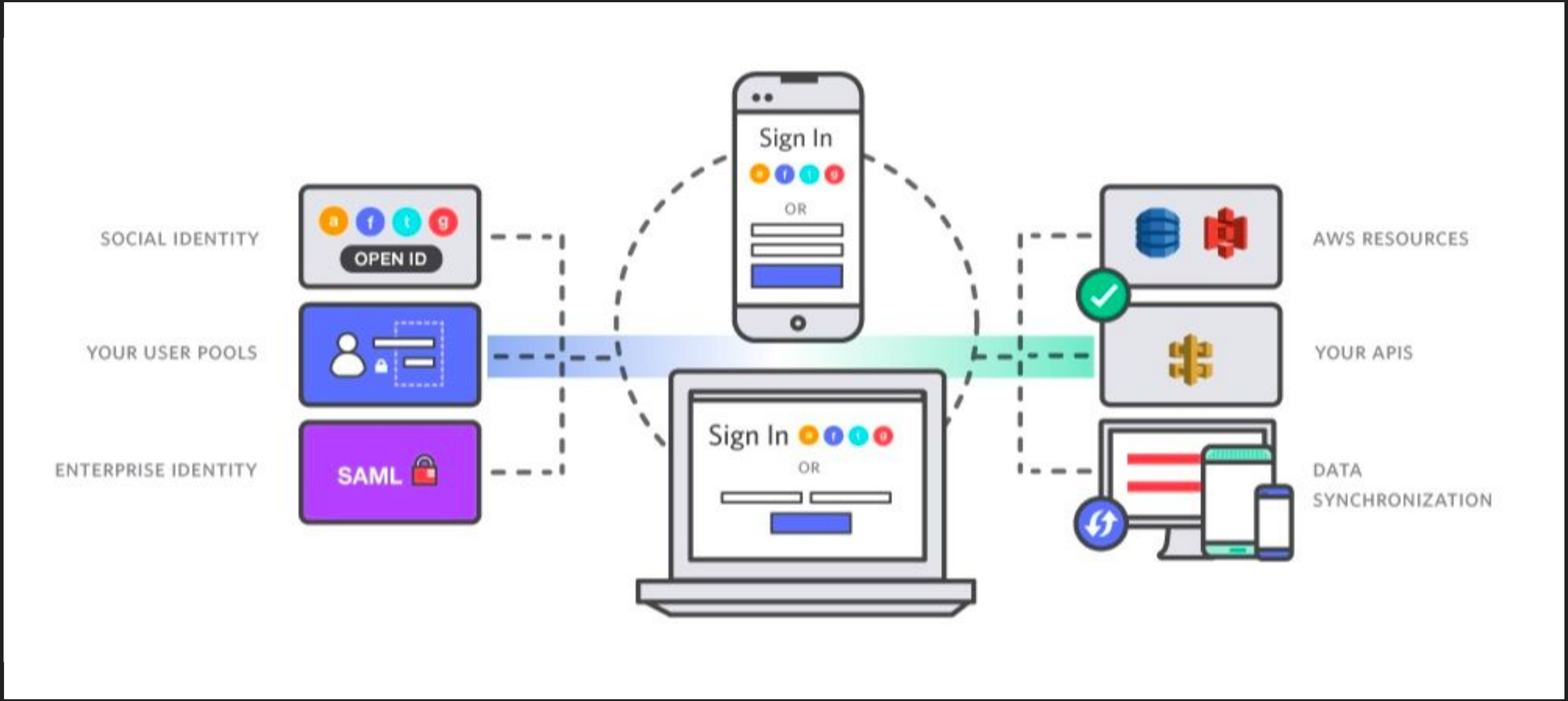
Our API is accessible by anyone on the internet.

IAM: Identity and Access Management

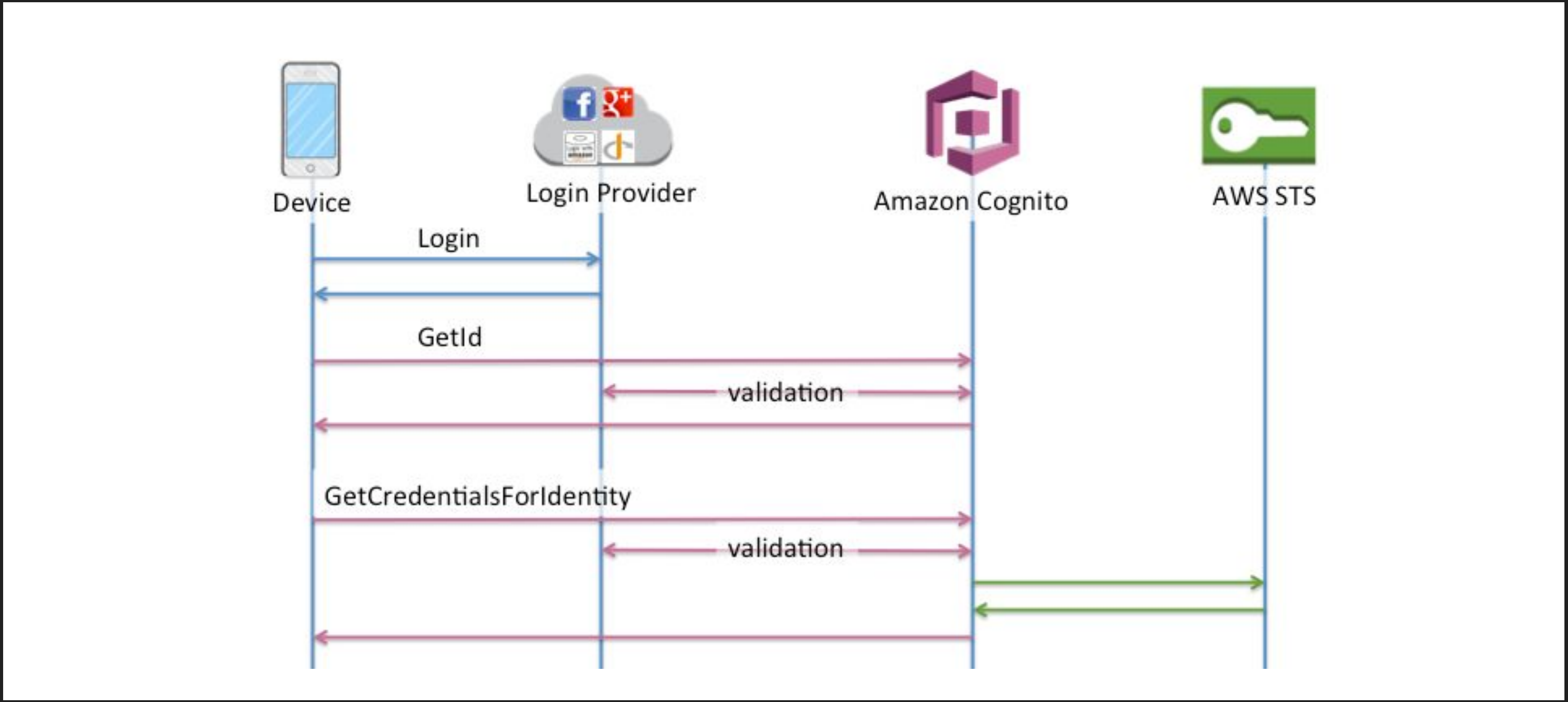
- Fine-grained access control to AWS resources
  - including API Gateway
- Create roles and permissions
- Integrate with your corporate directory
- Uses Access and Secret key pairs for access control
  - Can be used to sign API calls to AWS

Awesome! When was the last time you signed your API requests?

Cognito: User Pools & Identity Federation



Cognito: User Pools & Identity Federation



API Gateway Bonus: SDK Generation

- Takes a second to generate
- Support for multiple languages
    - Swift
    - Obj C
    - Java
    - Javascript, and more.
- Abstracts all the API calling complexity, including session signing

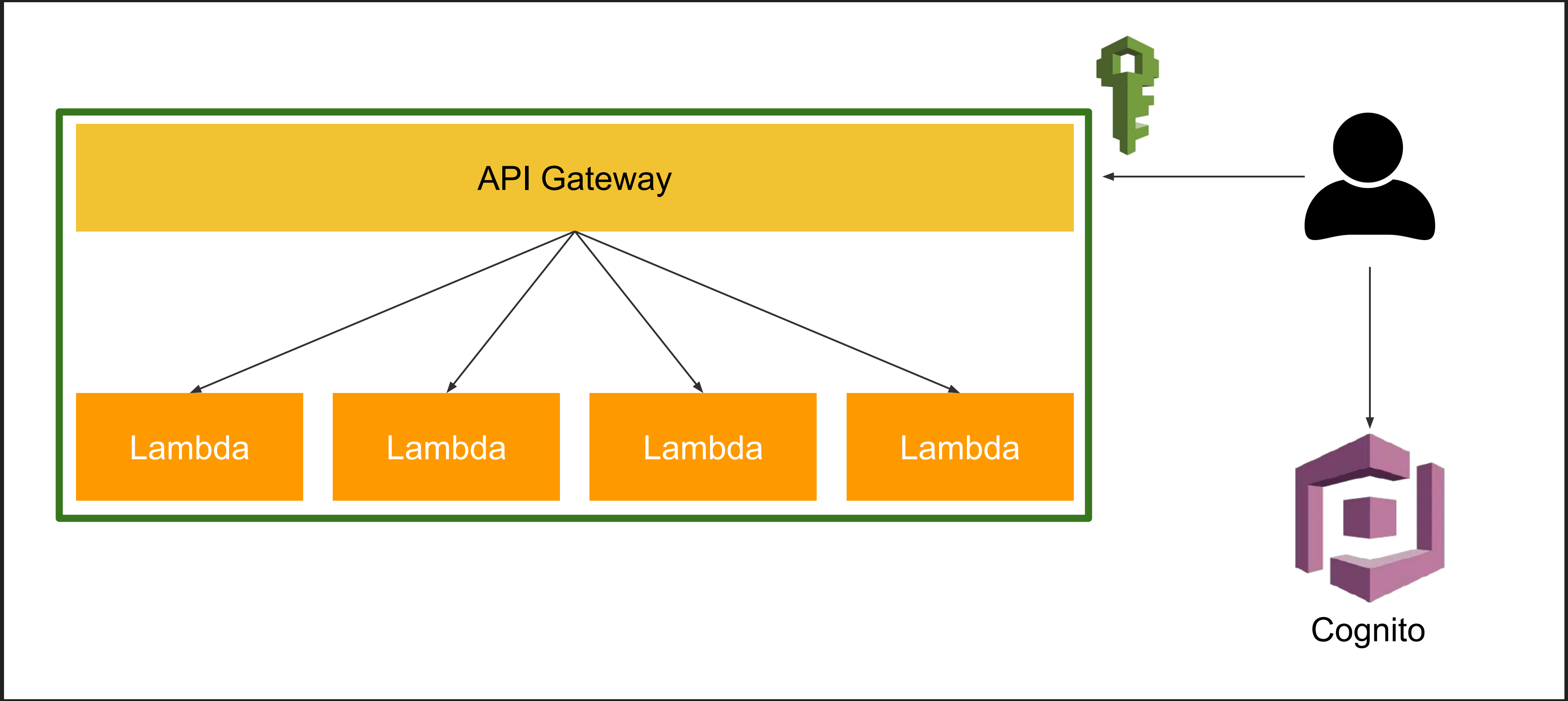
Cognito + IAM + API Gateway = Security [√]

- Cognito exchanges your session for temporary IAM credentials with limited permissions
- The API Gateway generated SDK signs API requests using the SigV4 signing process
  - Verify the identity of the requester
  - Protect data in transit
  - Protect against potential replay attacks
- Requests are executed with the caller’s credentials

DEMO

Integrate Cognito into the frontend application.

Now that’s looking a lot more secure!



Recap:

1. Use Swagger to design your APIs and documentation
2. API Gateway
    a. great API management tool
    b. seamless integration with Swagger
    c. generates SDKs for your API
3. Lambda
    a. serverless compute service
    b. integrates with API Gateway
4. Cognito
    a. useful for login workflows
    b. outputs temporary IAM credentials with custom permissions

Great, we have an API. What about the frontend?

1. Problems with Monolithic Systems
2. Microservices: A Solution
3. Microservice-driven APIs
4. Scaling the Frontend
5. Asynchronous Workflows

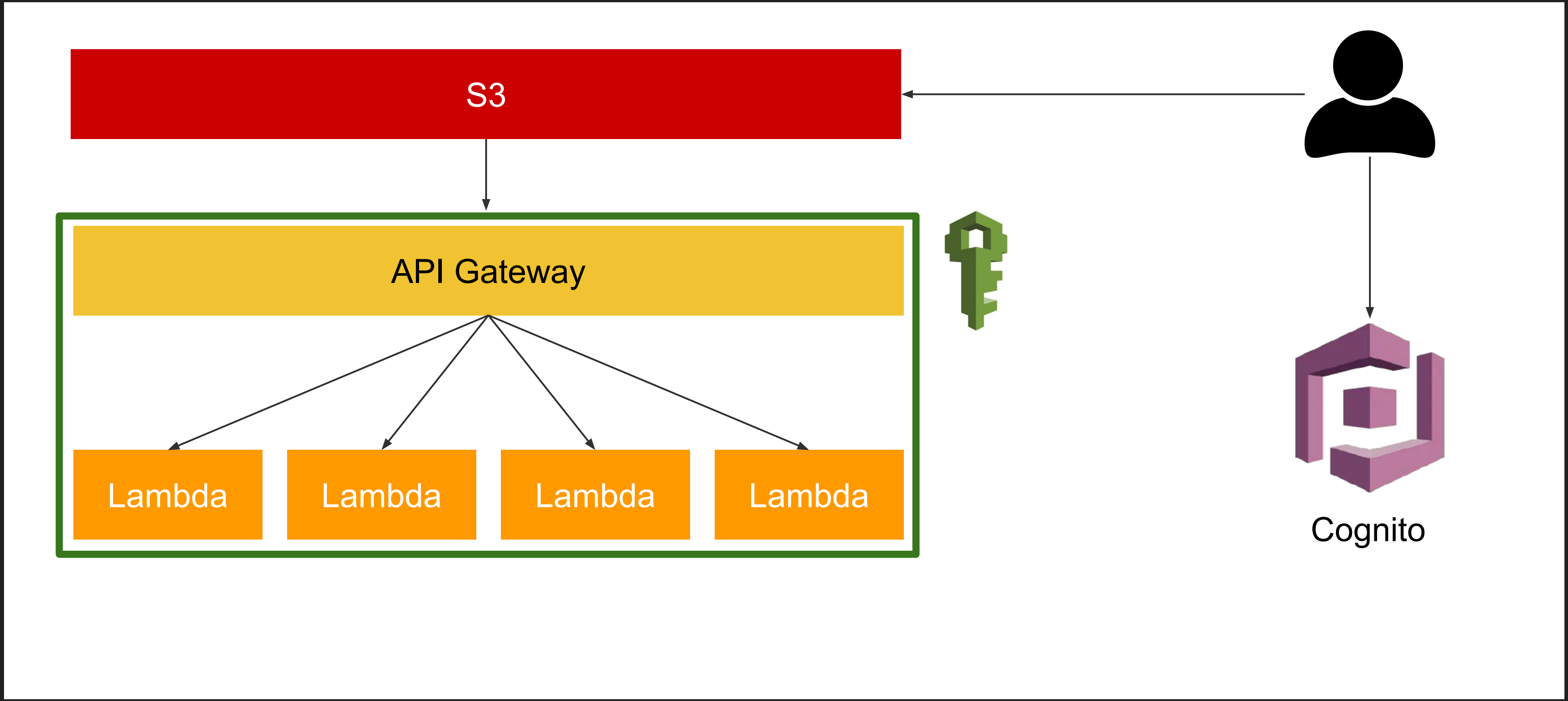
S3: More than just storage

- Host your website on S3
  - HTML, CSS, JS
- You get:
  - 99.999999999% of durability
  - 99.99% of availability
- You pay:
  - < $1 per year

DEMO

Host a static website on S3

One more revision of our stack



Recap:

1. You can use S3 to host your frontend
2. S3 hosted websites get out of the box scalability, availability, and durability

OK, let’s make some money.

You want to sell products through your concierge service.

1. Problems with Monolithic Systems
2. Microservices: A Solution
3. Microservice-driven APIs
4. Scaling the Frontend
5. Asynchronous Workflows

Current Checkout API

- Synchronous
- Overloaded
  - Performs checkout
  - Sends notification to user
  - (and in a proper implementation, it would also write to the db)
- More prone to failure

Asynchronous Checkout

- What?
  - process credit card transactions asynchronously
- Why?
  - defend against traffic spikes
    - 3rd party services are subject to downtime too
  - defend against programming errors and bugs
  - execute intricate order workflows, without impacting the user experience

The Asynchronous Toolset

SQS

The Asynchronous Toolset

SQS

Lambda

The Asynchronous Toolset

SQS

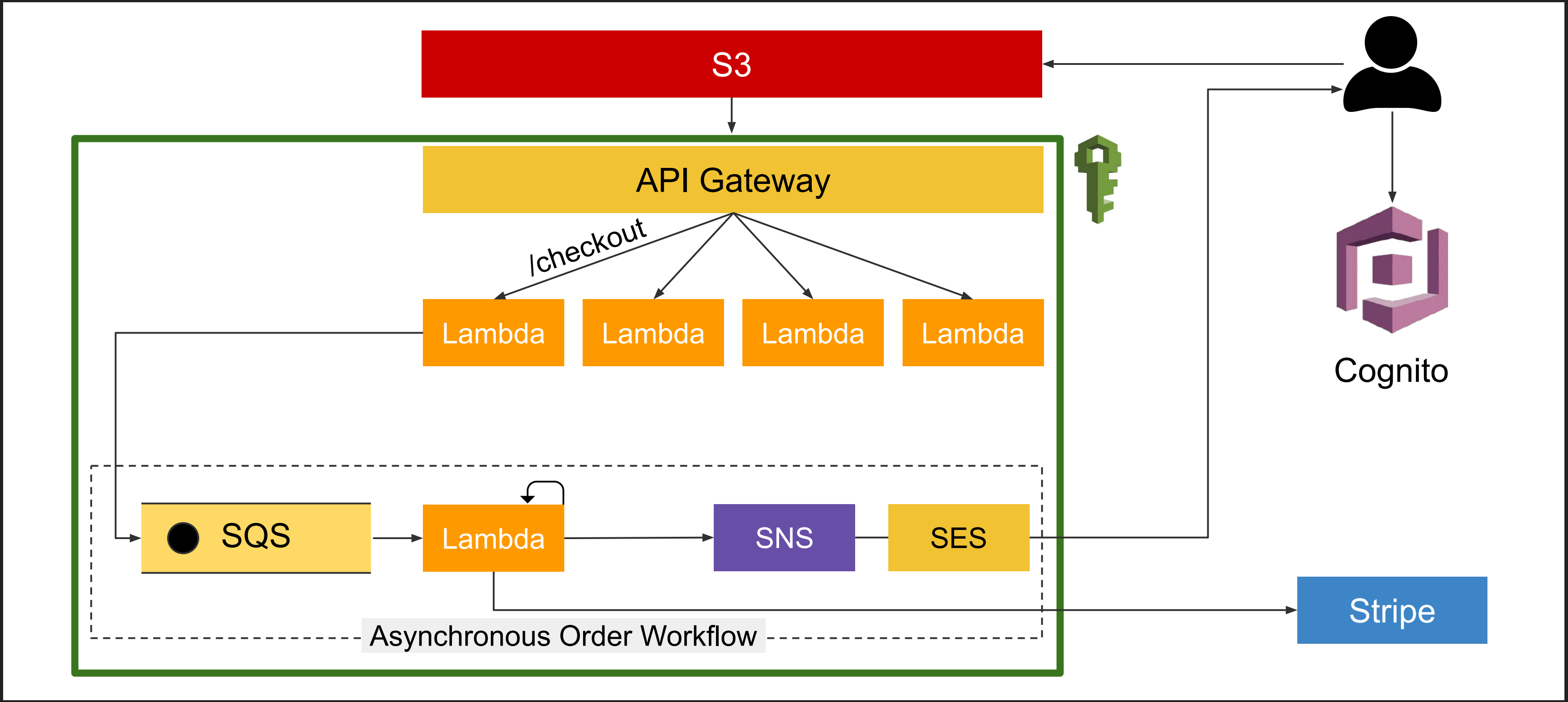
Lambda

SNS

DEMO

Setup the Asynchronous Checkout workflow

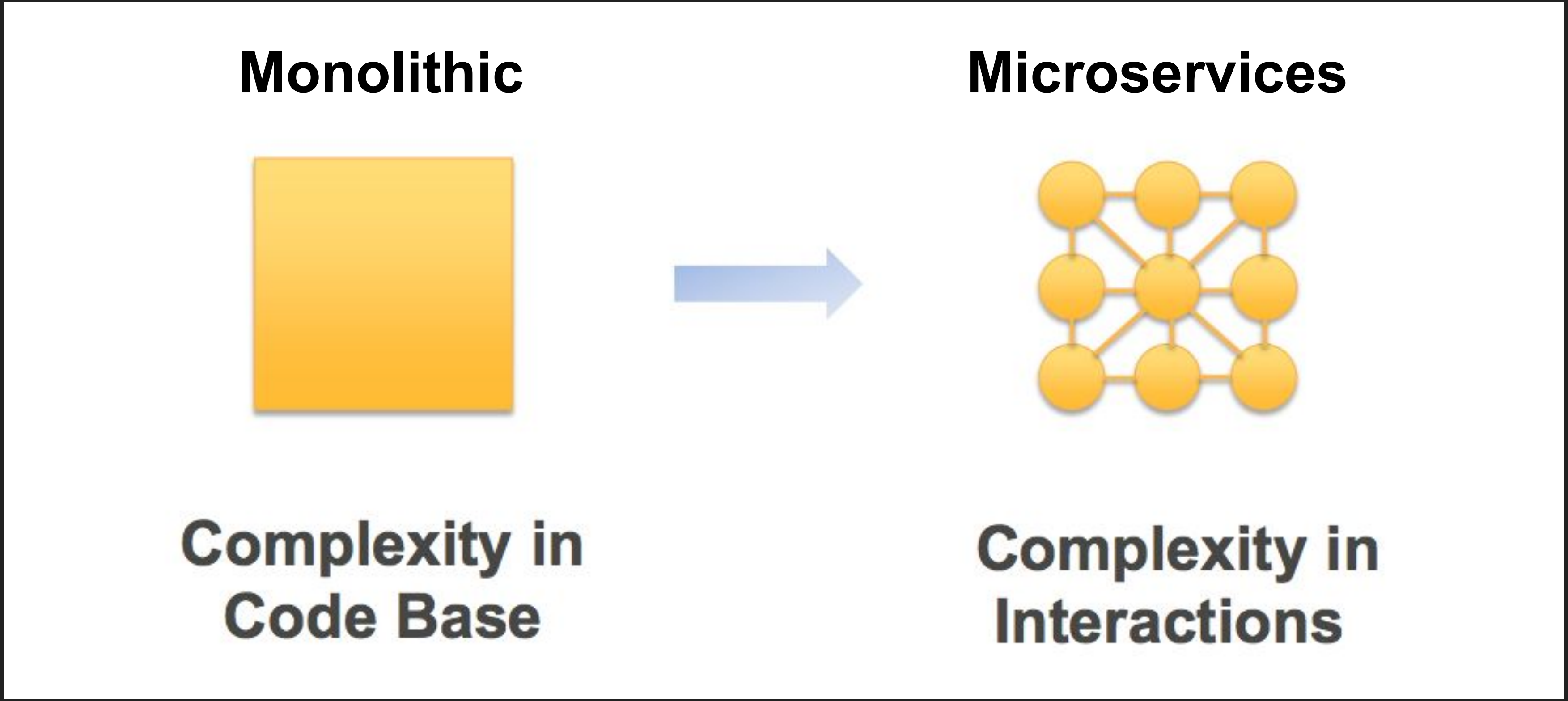
Another Stack Update



Recap:

1. Remove complex workflows from your APIs
2. Leverage SQS, SNS, and Lambda to distribute your application
3. Queues and notifications make your system a lot more resilient to failure

Microservices: Not a Panacea



THANK YOU!

Ideas for next steps...

1. Integrate a database (ex. DynamoDB) into the system.
   a. the database was omitted from the demo to emphasize how you can build a system service by service, rather than building everything at once
2. Integrate a distributed cache (Memcached or Redis) to store the NLU state between requests.
3. Use Route53 to setup the DNS for your domain.
4. Integrate a real NLP engine and the actual Stripe API to the overall system.

Sources:

https://aws.amazon.com/s3/
https://aws.amazon.com/iam/
https://aws.amazon.com/lambda/
https://aws.amazon.com/cognito/
https://aws.amazon.com/route53/
http://docs.aws.amazon.com/cognito/latest/developerguide/authentication-flow.html
https://aws-de-media.s3.amazonaws.com/images/AWS_Summit_Berlin_2016/sessions/pushing_the_boundaries_1300_microservices_on_aws.pdf

Codebase:

https://github.com/mangatanyc/columbia-lecture-concierge

Steps for building the frontend depicted in the demo:

1. download and setup bootstrap template [12.5 min]
   a. http://getbootstrap.com/examples/cover/#
2. create and integrate FB app [15 min]
   a. https://developers.facebook.com/docs/facebook-login/web
3. download and setup chat template [30 min]
   a. http://codepen.io/supah/pen/jqOBqp
4. create S3 bucket with static website hosting [5 min]
   a. http://docs.aws.amazon.com/AmazonS3/latest/dev/HostingWebsiteOnS3Setup.html
5. setup AWS profile using CLI [5 min]
   a. http://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html
6. deploy website to S3 [10 min]
   a. build a bash script that leverages the AWS CLI to upload your static files to S3
   b. https://github.com/mangatanyc/columbia-lecture-concierge/
7. generate Api Gateway SDK [1 min]
8. integrate Api Gateway SDK [15 min]
   a. http://docs.aws.amazon.com/apigateway/latest/developerguide/how-to-generate-sdk-javascript.html
9. setup Cognito Identity Pool [5 min]
10. integrate Facebook login with the Identity Pool [15 min]
    a. http://docs.aws.amazon.com/cognito/latest/developerguide/facebook.html