Tips and Tricks for Building and Testing Successful Cloud Applications and Services

Alex Bicalho

Infrastructure Automation Engineering Manager

FORGE DEVCON

AUTODESK UNIVERSITY

About the speaker

Alex Bicalho

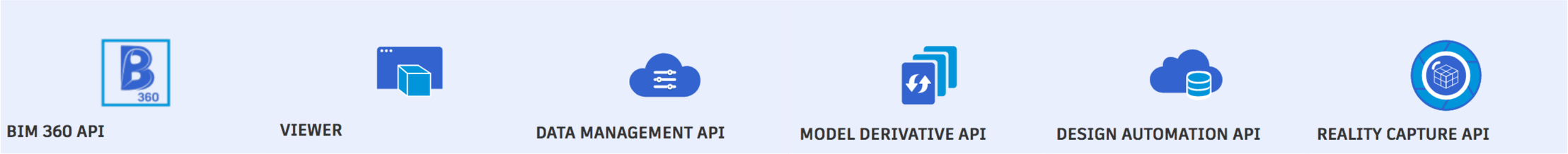
Alex has been working at Autodesk for 18 years. He worked as a QA Engineer in 3dsmax, AutoCAD and the Platforms Graphics SDK, later in the Cloud Platform team as a Test Engineering Manager and is most recently managing the Cloud Infrastructure Automation team. He believes in Automating all repetitive tasks; in building resilient, cost effective and infinitely scalable systems.

Topics

Product Software Development Lifecycle

- From Idea to Product
- CI/CD Pipeline
- Testing the code
- Building Resiliency in the Cloud
- Managing Data and Security
- You build it, you RUN it!
- Managing cost and making a profit

In the beginning…

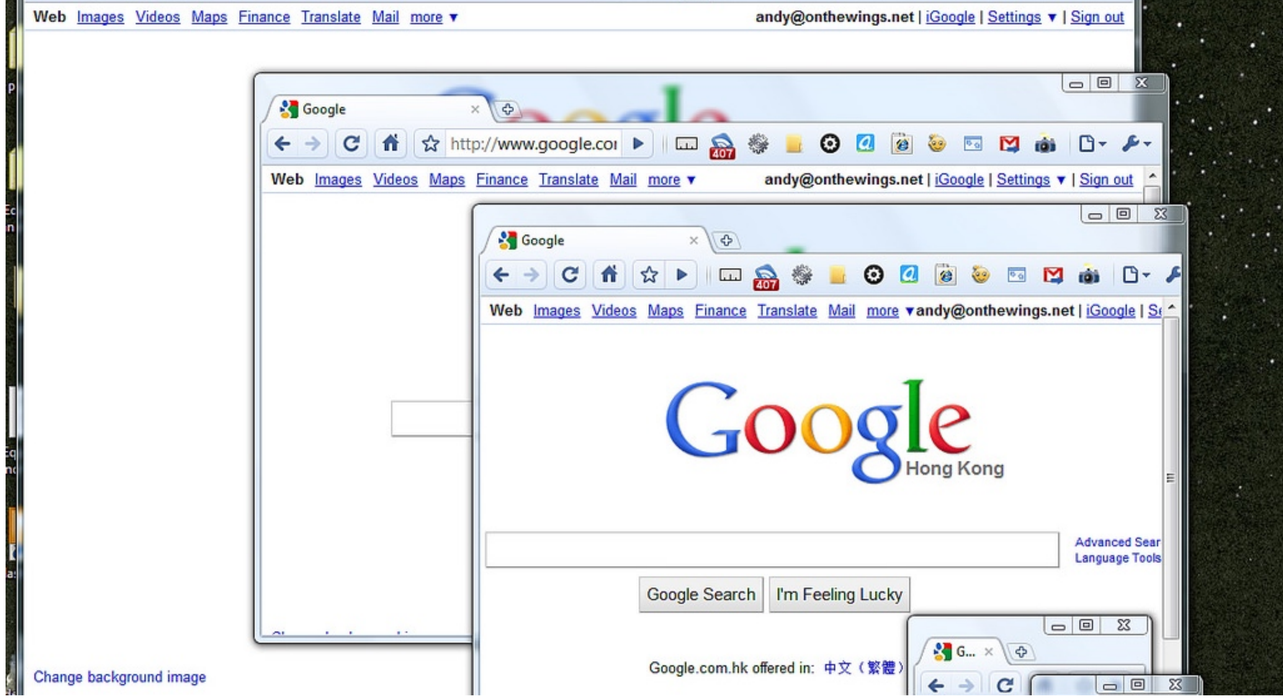


Autodesk Forge

Forge gives companies the tools to develop custom, cloud-based software applications that connect workflows for manufacturing, media/entertainment, architecture, engineering, and construction.

From Idea to Product

- Requirement Gathering
- Build a POC
- Think about Design and Architecture
  - Desktop or Mobile
  - Web or Cloud
- Build a better product
- Test the application with all dependencies
- Deploy, Run, Monitor



Desktop and Mobile

- Programming Language constraints
- Installation and licensing considerations

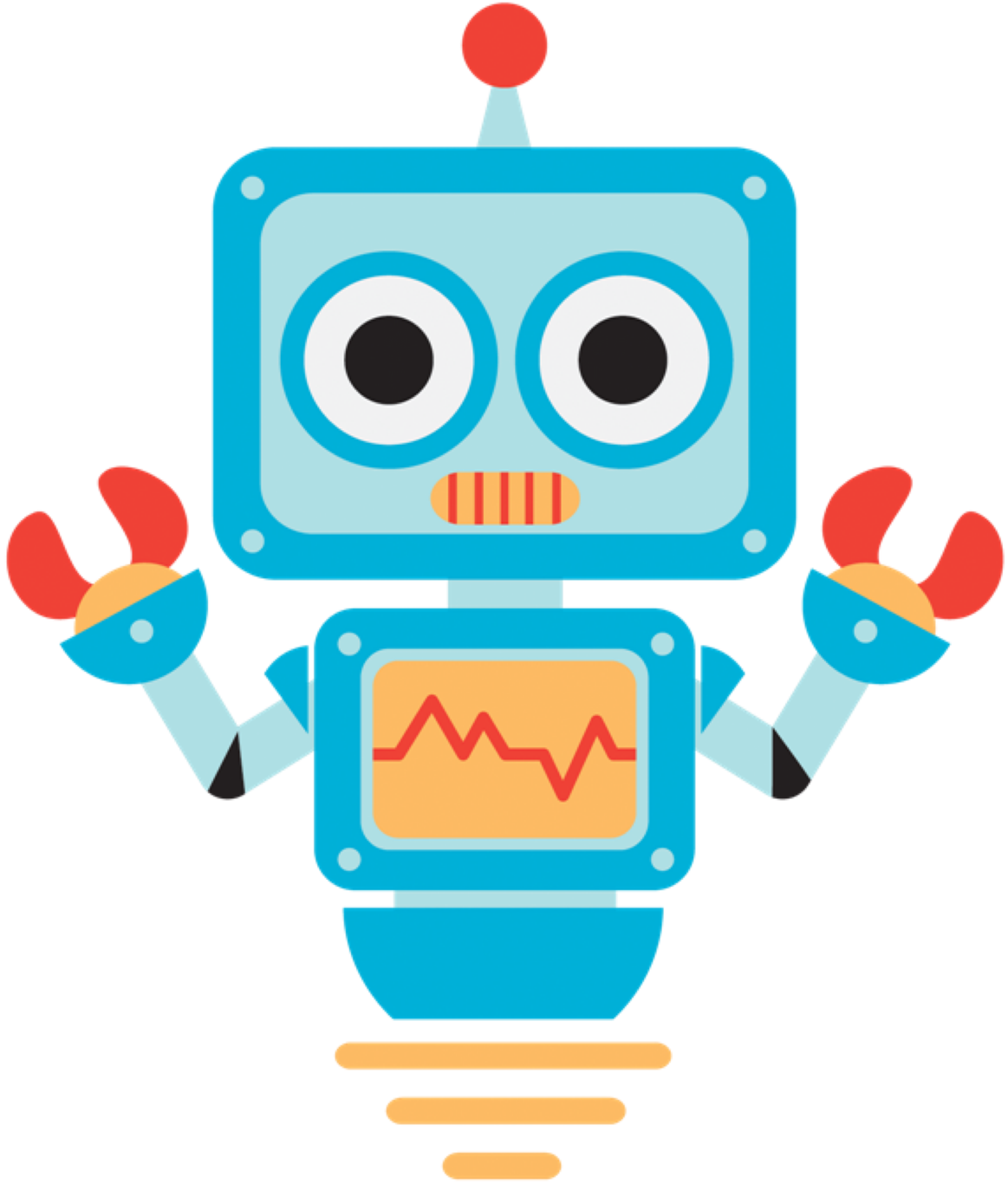
Web App and Cloud Services

- Runs on a Web Server on a data center
- Needs to be deployed, monitored, managed
- You’re responsible for the costs

Deploying Web Applications and Cloud Services

Automation!

- Use Tools provided by Cloud Providers or Cloud Agnostic tools
  - Cloud Formation or Terraform
  - Chef, Puppet, Ansible
  - Etc.
- Treat Deployment as source code



Deploying Web Applications and Cloud Services

The 21st Century is here!

- Use Containers
  - Pick the proper orchestration solution
  - Build lean containers
- Take advantage of Serverless technologies
  - Find the proper balance between server vs serverless

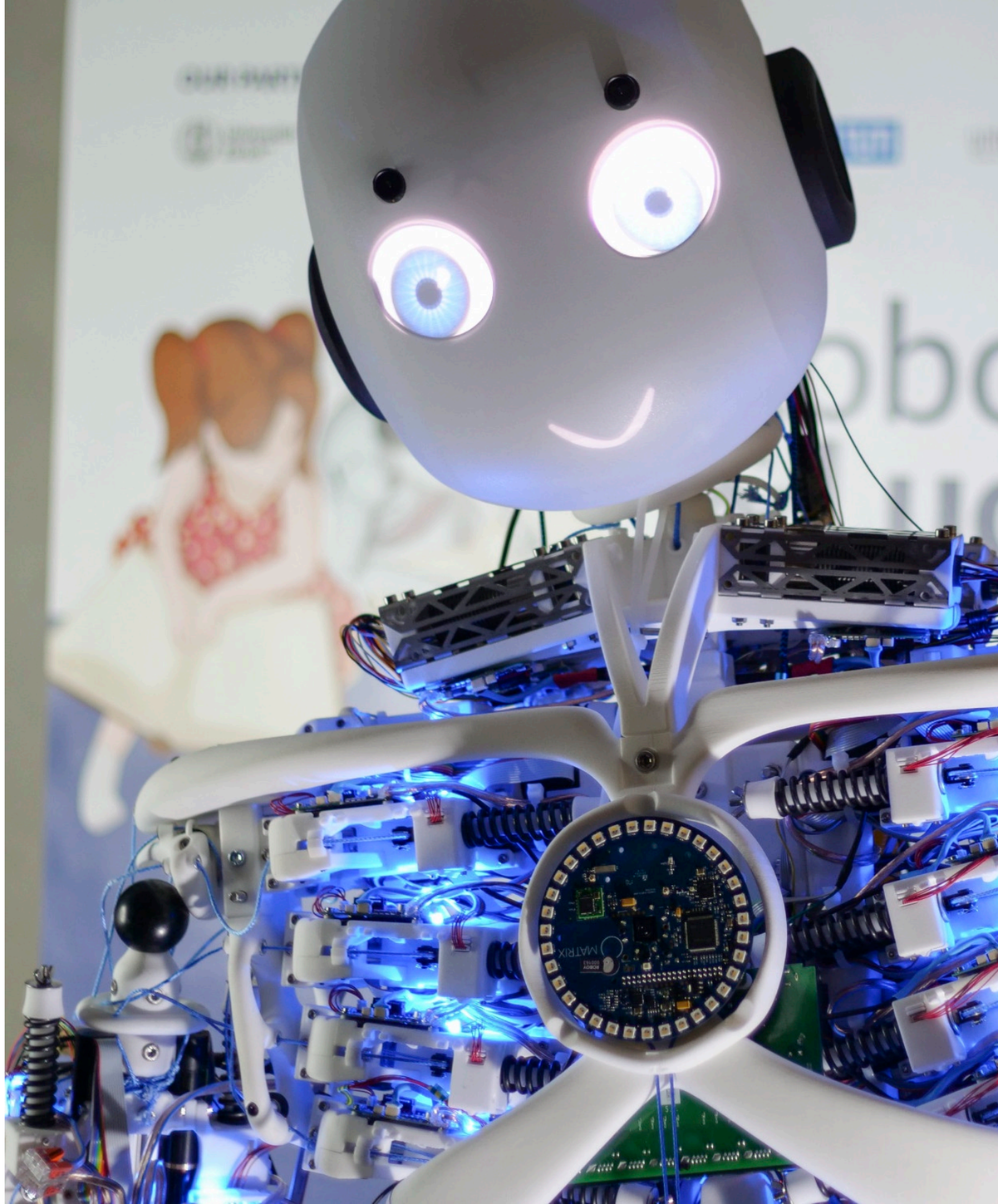


Image courtesy of Roboy

Setup a CI/CD Pipeline

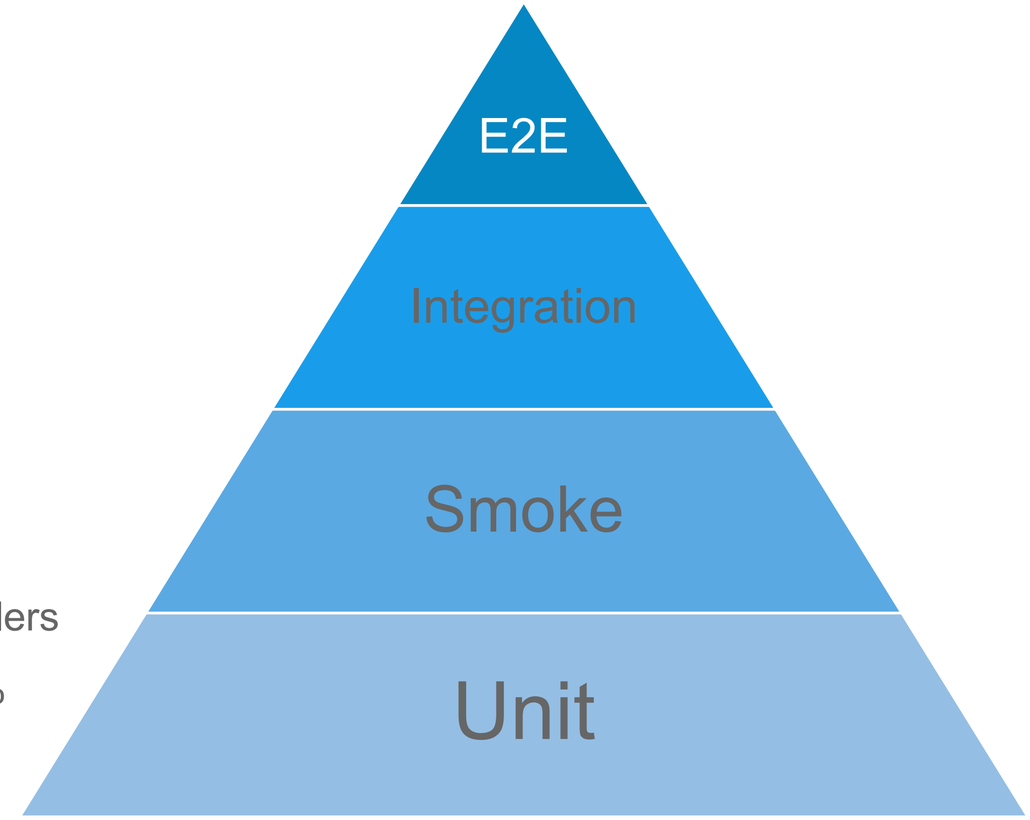
Repeatability!

- Setup a CI/CD Pipeline
  - Every code change committed should be deployed
- Use common tools or services
  - CircleCI, Travis
  - Jenkins, Bamboo
- Setup multiple environments (Development, Production)
  - Every commit deploys to Dev if tests pass
  - Releases go all the way to Production

Onto Testing!

Testing Strategy

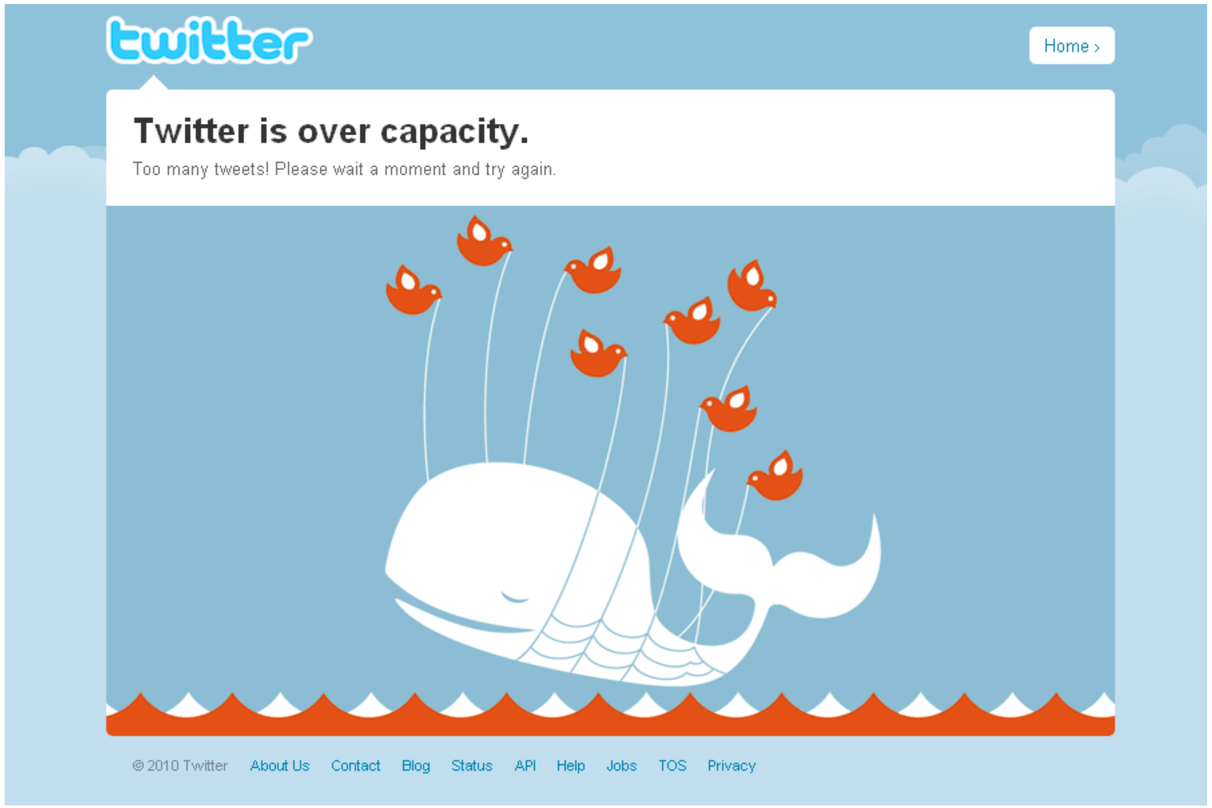
- Shift-left for Testing
  - Unit Tests to test the code being written
  - Use TDD to speed up development
- Architect for testability
  - No Business Logic in UI or Data (MVC – Model, View, Controller)
  - Separate the HTTP stack from the methods/classes
- Create “Smoke” or Acceptance Tests
  - Validates 100% of the REST APIs, or 100% of the controllers
  - Run these at every deployment/release and require 100% success rate
- Create fewer, but important integration or End-End tests



Testing a Web App or Cloud Service

- Networking Failures and Latency
- No control over the “hardware”
- Differences between OS, file management
- Scalability and Performance
- HTTP Error Codes
- String encoding and management
- JSON vs XML
- Resiliency Patterns

Network calls

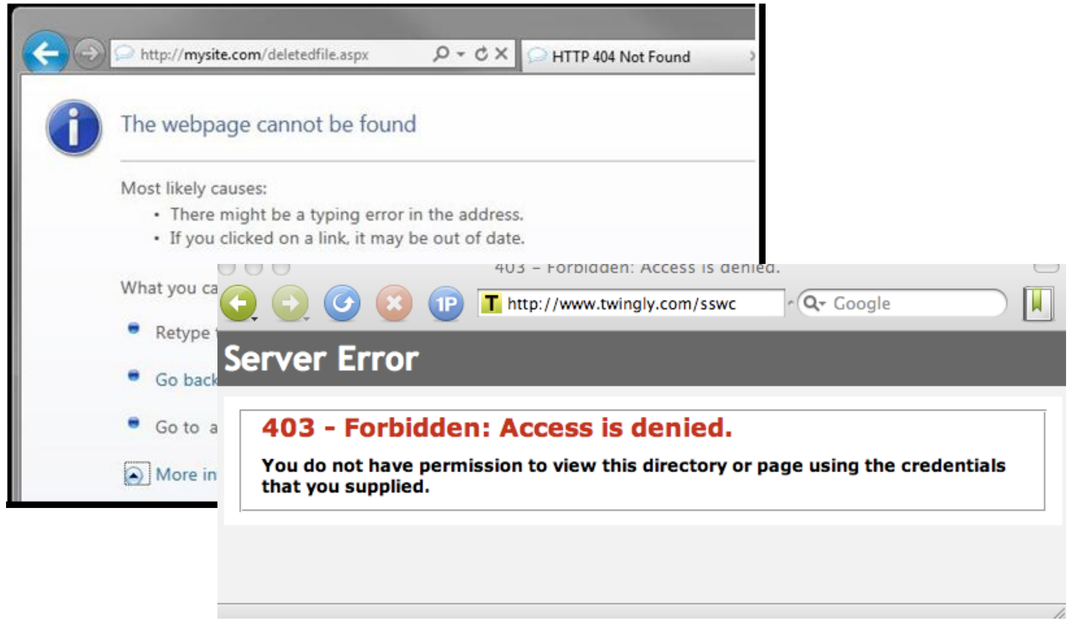


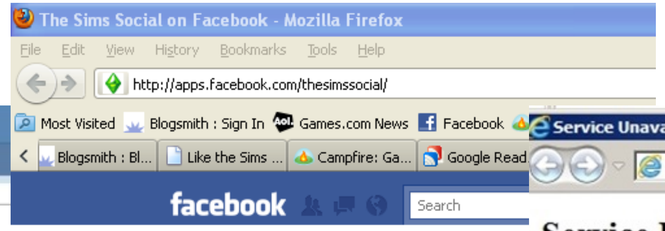
Network calls

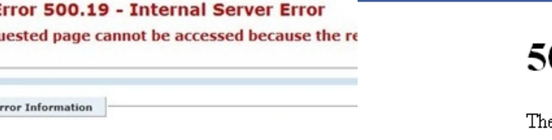
Net..orks fa..l onc.. in a whi..e …

- Consider how your application reacts when a failure happens
- Add testcases for network or dependency failures
- Add the proper response codes to notify your users when failure happens
- Handle HTTP Error code and timeouts appropriately
- When dealing with large content (files/videos):
  - Split uploads/downloads in chunks if supported
  - Use Hash validation
  - Paginate results when querying data stores

HTTP Error Code and Timeouts







HTTP Error Code and Timeouts

HTTP Response Codes

- As a Consumer:
  - How do you handle HTTP responses different than 200, 404?
  - Always retry with exponential back-off and jitter, and respect the Retry-After header value
- As a Producer:
  - Document your Response Codes and how your clients should handle errors and retries
  - Notify your customer about the reason for failure in a secure and clear way

Timeout management

- HTTP 504 means that the service took too long to answer
- Protect your responses for timeout by ensuring they finish or close network connections in a short time

No control over the hardware

“The Cloud is just someone else’s computer”

- Servers can (will) be recycled
  - Elastic auto-scaling
  - Containers are ephemeral
- Serverless compute takes time to be provisioned
  - First run takes longer

OS Differences

Windows vs Mac vs Linux(es)

- Registry
- File Systems (FAT/NTFS)
- Permissions
- Powershell/CMD
- High footprint, UI
- Specific Windows version for containers

- Config files in disk
- File Systems (many)
- Permissions
- Terminal
- Low footprint, no UI
- Different Linux distributions

String Management

- URL encoding
  - “Hi%20there” == “Hi+there”
  - Encoded string is larger than regular string
  - Double URL encoding issues
- URN, Base64 and other types of encoding
  - Multiple types of Base64 encoding patterns
- Character limits for URL, POST data
  - OS limitations dealing with Filenames

- Regular string:
  私の名前はアレックスです
- URL Encoded:
  %E7%A7%81%E3%81%AE%E5%90%8D%E5%89%8D%E3%81%AF%E3%82%A2%E3%83%AC%E3%83%83%E3%82%AF%E3%82%B9%E3%81%A7%E3%81%99
- Base64:
  56eB44Gu5ZCN5YmN44Gv44Ki44Os44OD44Kv44K544Gn44GZ

XML vs JSON

- XML can have Schema and Namespaces, JSON does not
- JSON allows for some constructs that are illegal as XML
  - "" : "something" is valid
- JSON and XML extraneous character handling is different
  - " vs %22
- JSON support in .NET isn’t as strong as in Java, JavaScript

Scalability and Performance

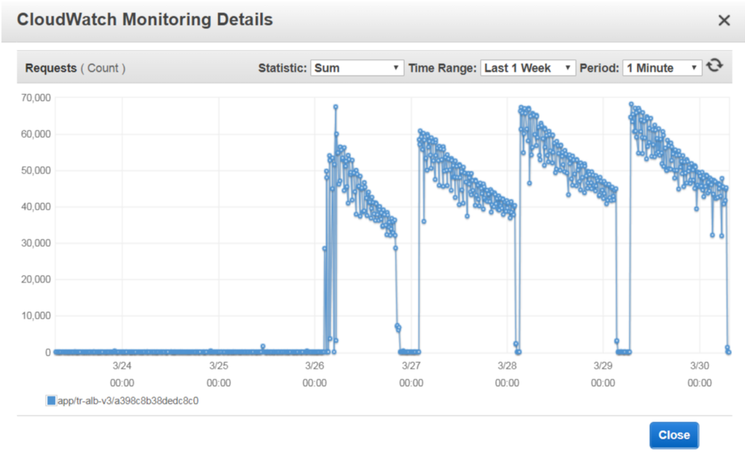
- Testing for individual server performance
  1. Build Load Tests
  2. Run the tests increasing the number of users against 1 server
  3. Identify the point at which the server “breaks”
  4. Use 80% of that number of users as your hard limit for elasticity and auto-scale

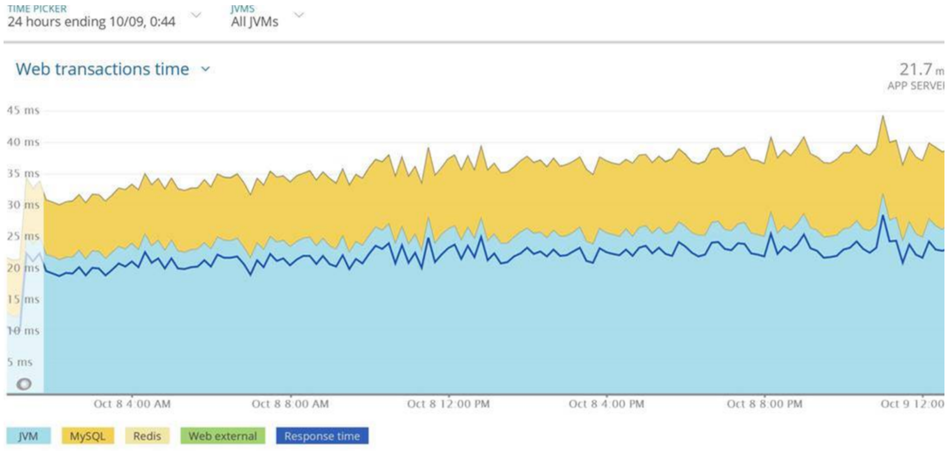
- Performance test
  - Validates the performance build after build

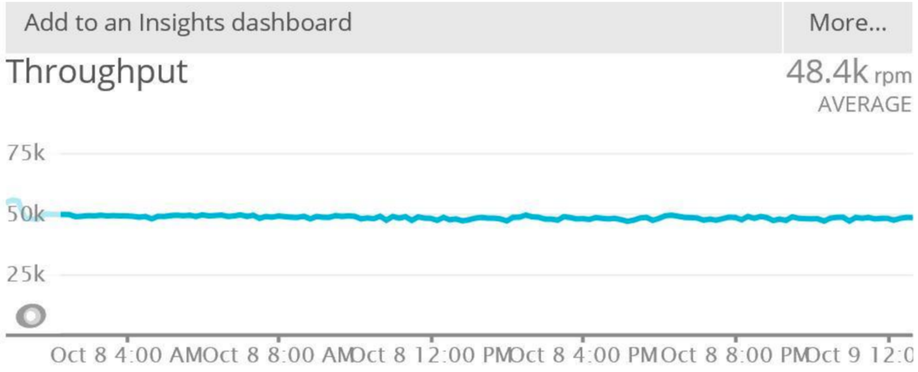
- Testing for elasticity
  1. Build Load Tests
  2. Setup an elastic cluster which increases capacity with the settings found in the individual server
  3. Run the tests increasing the number of users until 5-10x of the max number of users for 1 server
  4. Validate that new servers are added, that they respond timely and that the graph is linear

- Longevity test test
  - Validates the performance and stability over a long period of time (ex.: 24 hours, 1 week)

Scalability and Performance







Resiliency Patterns

- Handling failure
  - Circuit breaker
  - Queue Based leveling
  - Throttling
  - Etc.
- Designing Fault Tolerant UI
  - Mobile Applications with *Offline* mode

Tools and SDKs:

- Netflix Hystrix
- Polly
- Envoy

Handling data

- Dozens of Database types
  - SQL-like
  - No-SQL
  - Files
- Each DB for its own goal
  - Key-value
  - Time-based
  - Complex queries
- Database hosting
  - Provider managed
  - Self-hosted

Suggestions

- Use provider managed databases
- Build Business Logic in the code, not the database
- Avoid Joins
- Always use Encryption
- Limit the number of users, secrets for access
- Frequent backups stored on a separate account
- Test backup and restore

You build it, you RUN it!

Monitoring

INFRASTRUCTURE

Ensures that the virtual machine is healthy

- CPU, Disk and Memory usage
- Temp folder free space
- Other OS constraints (Registry, network connections)

APPLICATION

Ensures that the application is working as expected

- Error rates and response time
- Runtime health
- Log analysis and metrics

DATABASES

Ensure the databases are working as expected

- Infrastructure (metrics available)
- Query response time

USER

Monitors the application from the user point of view

- Synthetic tests
- UI-based analytics and monitoring

Managing application Costs

CLOUD COST STRUCTURE

- Pay per use
  - Per hour for Compute
    - CPU and Memory
  - Per byte for Data
    - Amount compounded month over month!
  - Per transaction for others

Suggestions

- Start with a small compute instance
- Use elastic scaling
- Consider a data cleanup strategy
- Optimize your code – you’re paying for every CPU cycle!
- Understand your application cost

Profit!

Summary

DEVELOPMENT

Define your application architecture

Setup automated deployment

Create a CI/CD

TESTING

Build tests early on

Architect for testability

Create a comprehensive testing strategy

MONITORING

Setup monitoring at every layer of the application

Learn about problems before your customers do

Ensure your customer has a positive experience

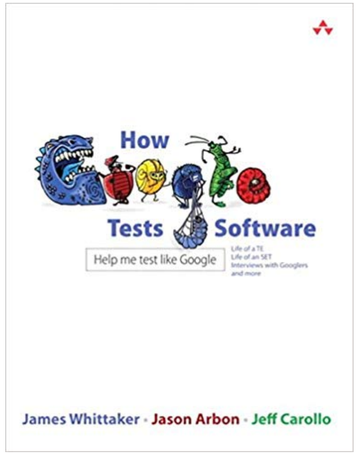
COST

Understand your application costs

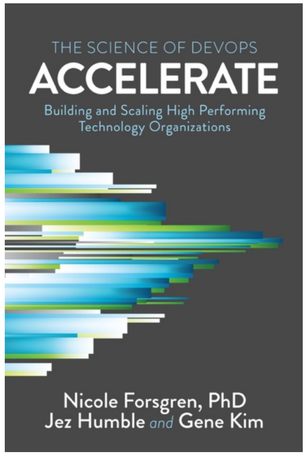
Charge your customers enough to turn a profit

Additional Resources

HOW GOOGLE
TESTS SOFTWARE



ACCELERATE



AWS WELL-ARCHITECTED

https://aws.amazon.com/architecture/well-architected/

Questions

AUTODESK®

Make anything™

Autodesk and the Autodesk logo are registered trademarks or trademarks of Autodesk, Inc., and/or its subsidiaries and/or affiliates in the USA and/or other countries. All other brand names, product names, or trademarks belong to their respective holders. Autodesk reserves the right to alter product and services offerings, and specifications and pricing at any time without notice, and is not responsible for typographical or graphical errors that may appear in this document.

© 2018 Autodesk. All rights reserved.