FDC225862

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

Alex Bicalho
Autodesk Inc

Learning Objectives

- Learn how to release code changes faster and with higher confidence
- Learn how to build a CI/CD for cloud services
- Learn about building test cases for cloud applications and services
- Learn how to build test cases for failure scenarios common to cloud applications and services

Description

Web Applications and Cloud Services are highly distributed and loosely coupled. They have dependencies on both internal and external services and are released on different frequencies and schedules. The distributed nature and variety of the Cloud ecosystem makes it so that teams have to think about testing differently. This presentation will cover tips and tricks to enhance your development lifecycle to release changes faster, with higher confidence; how to build a CI/CD pipeline; how to package and deploy your code; when and where to introduce testing in the process; and what testing should be done.

Speaker(s)

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.

Table of Contents

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.

Using Autodesk Forge

You can use the Forge Platform to offer your clients one of the following solutions:

- A Desktop or Mobile application
- A plug-in for a desktop or mobile application
- A Web application
- A Cloud Service

From Idea to Code

Let's say you have an idea and you want to put it in practice. Forge services allow you to quickly create proof of concept scenarios by interacting with the Forge Services using REST APIs in code, CURL, or using REST API tools like Postman (https://www.getpostman.com).

Software Development Lifecycle

A typical Software Development Lifecycle has the following steps:

- Requirement Gathering and Analysis
- Design and Architecture
- Development
- Integration and Testing
- Installation and Deployment

The difference between a Desktop or Mobile application from a Web Application or Cloud Service is that one needs to be installed on the user's device, where the other runs on a web server. All the concepts described in this presentation apply pretty much equally to both cases – in one scenario you need to deploy your application onto a web server running on a public cloud; in another you need to build, package and sign your application to be submitted for review by an App Store.

This presentation will focus on Web Applications and Cloud Services, but it will also provide you some tips and tricks for Desktop and Mobile apps.

A great write up on how to design good Web and Cloud applications can be found at the AWS Well Architected website - https://aws.amazon.com/architecture/well-architected

Software Design and Architecture

It's important to design the architecture of your application/service before you start implementing it. You should understand the user flow, the data flow, where data is stored, how your credentials are stored and managed, how your customer's licenses work, etc.

During the design phase you also choose your programming language, high level architecture, etc. There are multiple different programming languages, libraries, plugins and helpers you can use – each will be dependent on the solution you're designing, your targets, your workflows, etc.

It's important to visualize how your user will be using your application so you can design it appropriately. For instance, if you're designing a Mobile Application you should consider local caches, poor network connections, offline mode, etc. If you're designing a web server you need to consider API Rate limiting, dependency management, state management, etc.

Desktop and Mobile applications

If your solution requires you to build a Desktop or Mobile application, or a plugin for those, you'll be somewhat constrained in what you can choose for programming languages, installers, and other tools.

Mobile applications have their own installation and licensing schemas, so that's not something you need to worry about.

For desktop applications and plugins, you can use the same processes you used for other ADN applications or plugins.

Web Applications and Cloud Services

A Web Application or Cloud Service will run on a web server, most likely on a data center or a public cloud. As such, it needs to be installed and configured, a process called Deployment. Pretty much every programming language is supported one way or another on a Web Server – from Cobol to Erlang, from Lisp to Python.

There are various ways deploy an application on a web server, but it's important to do so through automated tools and scripts – this guarantees that all deployments yield the same result and are easily reproducible. Another upside is that all deployment or configuration changes are visible in the code.

Here are several tools recommended for deployment which are provided by different cloud providers:

- AWS
  - AWS Beanstalk - https://aws.amazon.com/elasticbeanstalk/
  - AWS ECS for Containers - https://aws.amazon.com/ecs/
  - AWS Container service for Kubernetes - https://aws.amazon.com/eks/
  - AWS CloudFormation - https://aws.amazon.com/cloudformation/
- Google Cloud Platform
  - Developer Tools - https://cloud.google.com/products/tools/
- Microsoft Azure
  - Azure Developer tools - https://azure.microsoft.com/tools/

There are tools which are provider agnostic and allow you to deploy your application or service in different providers:

- Terraform - https://www.terraform.io/
- Spinnaker - https://www.spinnaker.io/
- Chef - https://www.chef.io/
- Puppet - https://puppet.com/
- Ansible - https://www.ansible.com/
- SaltStack - https://saltstack.com/

Using Containers or Serverless technologies

In 2018 we cannot discard Containers and Serverless applications in our portfolio.

Containers are an excellent way to package all dependencies into a single object that you can then simply deploy and run very quickly.

Serverless applications are just what the name says – there's no server for you to manage, you simply upload your container or code onto the cloud and it just runs based on the rules you set.

There are several advantages of Serverless applications – the main one simply the fact that you do not need to manage the infrastructure.

One disadvantage of the serverless application is that it's designed for transient compute, meaning, it's designed for short-lived tasks. For instance, an AWS Lambda will only run for 15m maximum. If you need to run a task that is longer than 15m, then you should consider a webserver or a serverless container.

It's also important to consider the proper data management architecture behind a serverless architecture, because the functions or containers will be stateless and will be recycled as soon as they've completed their task.

Putting it all together in a CI/CD Pipeline

The faster you can get your software to your client, the sooner you will reap the benefits. How can you write software, make changes or release new features, and release them to your customer as soon as possible?

One of the best ways to do so is to setup a Continuous Delivery system – also known as Continuous Integration and Continuous Deployment system – CI/CD.

Assuming you are hosting your source code in github, you can use one of several CI/CD tools in the market:

- Circle CI: https://circleci.com
- Travis CI: https://travis-ci.org
- Jenkins: https://jenkins.io
- TeamCity: https://www.jetbrains.com/teamcity
- Bamboo: https://www.atlassian.com/software/bamboo

Most of these tools will allow you to build, test and deploy your software. Some are free, others are commercial; some are hosted solutions, others will allow you to deploy your own infrastructure.

Cloud Providers also offer CI/CD Tools as part of their tool portfolio:

AWS - https://aws.amazon.com/products/developer-tools/
Azure: https://azure.microsoft.com/en-us/services/devops/pipelines/
Google Cloud: https://cloud.google.com/solutions/devops/

Testing your application

A common goal is to release our application or service as soon as possible to our customers, and to make sure it's a high quality and useful application. For that reason, it is important to test our product.

Most modern programming languages allow you to define and run Unit Tests side by side with the code. This ensures that code is tested as it's built. A good practice is to build tests before writing the code – and it ensures that the code that is written works as it is expected because the application won't build if the tests do not pass.

Tests are organized in different test suites, and each test suite has a different objective. Here are the most common ones:

- **Acceptance or Smoke Test:** these are quick tests that are executed any time a product is deployed or installed to validate that the most common workflows work as expected.

- **Functional Tests:** these are comprehensive tests that validate the complete functionality of the application
- **Performance Tests:** these validate the performance of the application, and usually compares it against previous releases to ensure that no degradations were introduced
- **Capacity and Scalability Tests:** these tests are common for web applications and cloud services. They validate that an application or service is capable to work with multiple concurrent services and that more concurrent users can be added if the infrastructure is elastic and scalable.
- **Resiliency Tests:** these tests introduce fault injection on web applications and cloud services to validate how the application or services react to errors, network failures or when a dependency service or resource is not available.

Important considerations for Web and Cloud Testing

A Cloud Service or Web Application runs on a web server somewhere. It's accessible through the internet and relies on networking to communicate to its dependencies. This is the first and most important area to consider for failure and testing. Imagine what would happen if an application like AutoCAD had every one of the DLLs that it loads somewhere out there on the network instead of your local hard drive. That's exactly how the Web and the Cloud work!

As a rule of thumb, you should expect that at least 1 network call out of 10,000 will fail. In a DLL world, that would mean an application crash, unless you were expecting failures.

Other areas to consider

There are several other areas to consider when designing, deploying and testing your web application or cloud service:

- OS differences (Windows, Mac, Linux, different Linux distributions)
- String management
    - Character encoding
    - String size, limits
- HTTP Response Codes
- Handling Timeout and Retries
- XML vs JSON
- Concurrency, Chunking, Open Connection count

Load Testing REST APIs

Most of the Web -> Service and Service -> Service communication nowadays is done through REST APIs. Year after year it has become easier and easier to design and run Load Tests for those APIs. You can use these tests to validate the performance, capacity and scalability of the services.

These are various open source tools that you can use to design and run Load Tests:

- Apache JMeter: https://jmeter.apache.org
- Locust: https://locust.io
- K6: https://k6.io
- Taurus: https://gettaurus.org
- Gatling: https://gatling.io

If you don't want to setup your own test infrastructure, you can use one of these companies to run your Load Tests:

- Blazemeter: https://www.blazemeter.com
- Flood.io: https://flood.io

Building resilient applications and services

The internet is unpredictable, almost stochastic, mainly because of the dependency on multiple services communicating over the network. If you consider that your customer might be using a poor-quality cell phone in a rural area, or far away from a cell tower, you can imagine the quality of service they're getting.

On the upside, this problem is not new, and it's not unique to you – many other companies have already thought about it.

When building your application, consider using an SDK for fault tolerant network communications, like Hystrix developed by Netflix - https://github.com/Netflix/Hystrix. Hystrix was first written in Java, but it has since been ported to other languages.

When designing the UI you should consider these 10 usability heuristics to check which of them is important or necessary for your workflows: https://www.nngroup.com/articles/ten-usability-heuristics

If you're building a service, make sure your API documentation contains instructions for how your customers should react to failures and errors. For instance, should they retry on HTTP 5xx, and if so, how?

Build testcases that will inject failures in your dependencies, so you can test what happens when one of your dependencies has latency or failures. These are the scenarios you should consider:

- HTTP Response errors – 4xx, 5xx
- Network Latency and Timeout
- Service failures (databases, dependencies)

You can inject failure using a proxy server:

- Wiremock - http://wiremock.org/

- Betamax - http://betamax.software/
- Flashback - https://github.com/linkedin/flashback

You can use an open source tool or a Chaos Testing Service:

- Netflix Chaos Monkey: https://github.com/Netflix/SimianArmy/wiki/Chaos-Monkey
- Gremlin: https://www.gremlin.com

If you're testing on your desktop, you can use one of these applications:

- Fiddler - http://www.telerik.com/fiddler
- Charles Proxy - https://www.charlesproxy.com/
- Zed Attack Proxy - https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
- Network Link Conditioner (Mac) – installed as part of XCode

Data storage and security

It's very likely that your application will require you to store some data. You'll either need to store files or use a database to store data (or both!).

As I've showed in other examples, Cloud Providers make our lives very easy. They offer a myriad of services to manage data. It's much easier to use one of their database services, or to use one of their file storage services, than to purchase and manage your own. Their services will have high quality, are always up to date and have an extremely high availability.

It's very important that we use these services in a secure way, by properly segregating our customer's data, making use of data encryption, as well as making sure the database is only accessible within the private network, and only available to the specified services.

Your products will also need keys to access other services, or to decrypt data. You should make use of the secret management services that your cloud provider offers to do so. Besides storing keys and secrets, these services also offer tools that rotate secrets automatically, which is a best practice.

Monitoring the application

Your application is up and running on the cloud and your customers are happy, congratulations! What's next? Web Applications and Cloud Services run 24x7 on the cloud, this means they need to be of high quality and error free. These apps also need to be monitored so that you can resolve any issues that may arise.

Monitoring and Analytics

To monitor your application, you need to consider at least 3 different scenarios:

- System – is the infrastructure healthy, does it have available storage space, are CPU and Memory available?
- Application – is the application working as expected, or is it reporting errors or problems?
- Users – can your users use the application correctly?

Public Cloud Providers offer several tools for monitoring the infrastructure and application. Besides these tools there are several open source and commercial tools to monitor and analyze the application. Here are some of them:

- Prometheus - https://prometheus.io/
- NewRelic – www.newrelic.com
- DataDog – www.datadoghq.com
- App Dynamics – www.appdynamics.com
- Dynatrace – www.dynatrace.com
- Splunk – www.splunk.com

Besides monitoring your application it's also important to understand how your customers are using it. This analysis can be done with the tools above, or by using a Web Analytics tool like Google Analytics or Mixpanel. These will help you understand how your customers are using the application, if they're facing any issues or problems.

Understanding and managing the cost of your application

So your application is running on the cloud for 1 month and you receive your bill…

It's so easy to think that the Cloud is "cheap" because we often see the prices as cents/hour, but what we often ignore is the fact that there are 720 hours in a month. Likewise, when we look at the storage costs, they are GB per month, but if you do not have a storage policy and never delete or archive stale data, you'll be paying for it month after month.

As you design your application you should consider the number of servers and their size, the database and storage size, as well as understand how these resources can increase capacity when the number of users increases.

If you're using Compute resources, you should always design them to be elastic, and you should use auto-scaling groups. It's better to design nimble applications that run on small instances that can burst and scale up, rather than designing them with large server requirements.

Conclusion

When building Web Application and Cloud Services it's important to consider that you're not just building the product, you're also running it and paying for the cost to run, monitor and support it. There are several moving parts between the developing and deploying the application, and automated testing and deployment will greatly improve the speed and quality of development so you can get your product out to the customer in a very short time.