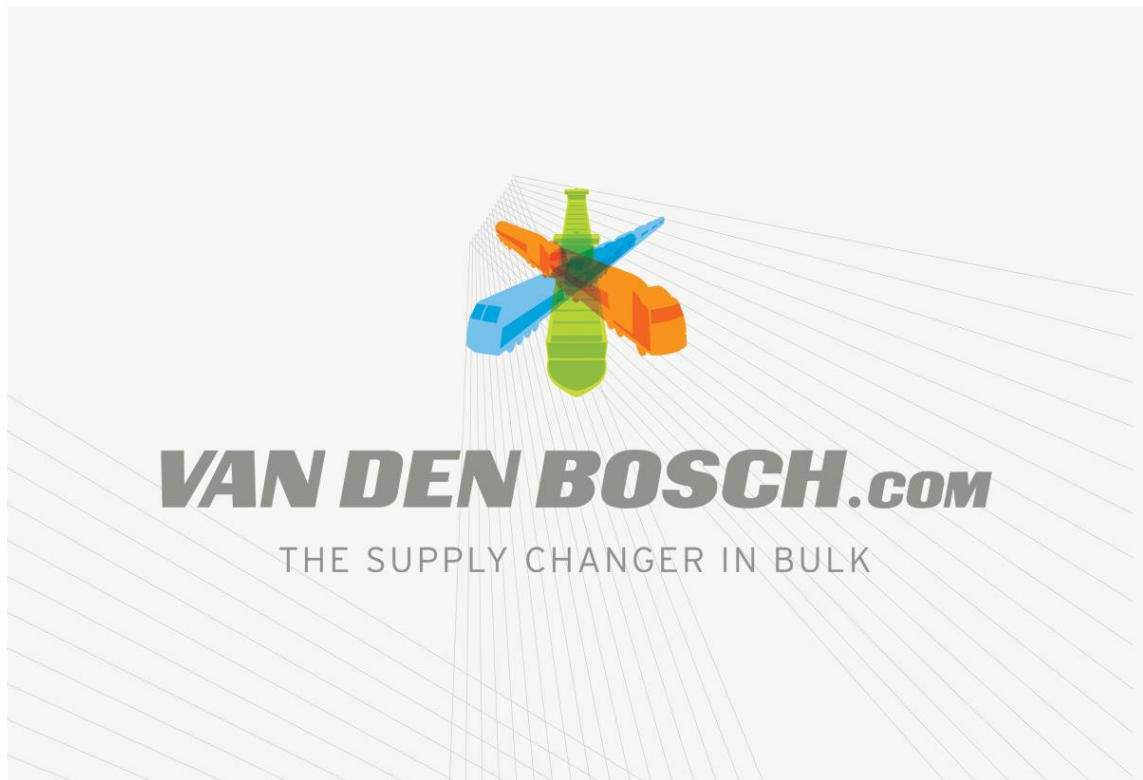


APRIL 13, 2023



# AZURE A/B TESTING RESEARCH

## HOW TO SET UP A/B TESTING IN AZURE ENVIRONMENTS

ROBIN VAN HOOF  
VAN DEN BOSCH  
Hoogven 10 Erp, 5469 EM

## Table of Contents

Introduction.....	2
Methods .....	3
What is A/B Testing exactly? .....	3
How do we set up an A/B Test? .....	3
Implementing A/B Testing in Azure.....	4
Preparing the versions for A/B Testing .....	4
Deploying versions .....	8
References.....	9

### **Versions**

Version	Date	Author(s)	Changes	Status
1.0	11-04-2023	Robin van Hoof	Setup, introduction, references	Work in progress
	13-04-2023	Robin van Hoof	Methods	Finalized version 1.0
1.1	13-04-2023	Robin van Hoof	Process proofread feedback	Finalized version 1.1
1.2	13-04-2023	Robin van Hoof	Add Dependency Injection provider	Finalized version 1.2

### **Distribution**

Version	Date	To	Goal
1.0	13-04-2023	Evalynn Kootstra – Fellow student	Proofread
1.1	13-04-2023	Luke van den Doelen – Internship Coach	Delivery

## Introduction

A/B Testing is one of the most common methods of evaluating the result of the implementation of a feature, most commonly used for frontend related features. A/B Testing is based on making two versions of an application, one with a feature implemented one way, and one without said feature implemented or implemented in a different way. The userbase will then be divided among the two versions of the applications and Key Performance Indicators (KPIs) will be measured to determine the performance of each version of the application.

Van den Bosch and Bulkio are looking for way to set up A/B Testing environments for their frontend applications. These applications are frontend Angular applications run in their Azure environment. This research will investigate how A/B Testing can be setup within Azure to measure how successful the implementation of a feature is and how users experience these features.

This research will focus on the deployment of A/B Testing in azure environments for frontend applications, mainly focus on applications developed on the Angular framework as this is the standardized framework within the company.

## Methods

### What is A/B Testing exactly?

A/B Testing, in the future referred to as ABT, is a form of testing the implementation of, usually frontend related, features and how users experience these features compared to a different implementation of said same feature. ABT works by making two versions of an application each implementing a feature in a unique way, deploying both applications and spreading the userbase over the two versions, measuring KPIs as they use the applications. This data is then gathered, and conclusions can be drawn from it.

This data is often collected in one of two ways: either measuring certain indicators like time spent on the site or asking user for direct feedback. No matter which collection type is used, the data is collected, tagged with the version of the application and analysed appropriately. From this data conclusions can be drawn on what implementation of a feature is more desired or better received by the users.

### How do we set up an A/B Test?

To set up an A/B Test it is important to first decide what feature we are going to test. ABT is effective at testing **one** feature at a time and one feature only. Theoretically testing more at once is possible but doing so could cloud the data and result in incorrect conclusions being drawn due to correlation between features.

Up next two version of the application need to be made. Often, ABT is done by using only two versions, A usually being the standard, already in use application, used as control group. B is usually a variation of A with one feature implemented in a way to be evaluated. However, ABT is also possible with more than two versions. In this case the approach changes from 'How well is this feature received?' to 'which way of implementing this feature is received best?'.

Then, an analysis needs to be made on what variables can be tracked to measure the success of a version. A few common ones are either clickthrough-rate (how many times a certain page is visited), time spent on page, or how many times an action is performed. These variables need to be tracked and labelled in some form. This will be talked about in chapter 'Implementation of A/B Testing in Azure'.

Now the stage is set to run the A/B Testing experiment. The multiple versions of the application will all need to be deployed and traffic will have to be randomly routed to each version. There are multiple ways to do this, usually unique to the deployment platform the applications run on. How this is done in Azure will be talked about in the chapter 'Implementation of A/B Testing in Azure'. Often a 50/50 or equal load between each version is used, however an equal split of load is by no means needed. Certain situations call for uneven distribution of load, for example when one version is not proved to be stable yet, thus reducing the possible impact of said instability by allowing less users on this version.

Each user is routed to one of the versions of the application and the variables set in the earlier step are tracked and collected. From this point on no more active involvement is needed from the testing team. A timeframe needs to be set between which the A/B Test will be run, usually spanning multiple

days or weeks as A/B Testing is not an instantaneous way of testing and needs time to accumulate enough data to draw useful conclusions.

After this timeframe is over and the test is considered finished, the collected data needs to be analysed in a proper way. The way this analysis needs to be done is highly dependent on what variables are tracked for the feature and guidelines are thus hard to set up.

### Implementing A/B Testing in Azure

Now the question remains of how we set up A/B Testing in Azure. Azure is chosen as this is the primary hosting service for the company, thus no new services need to be acquired, set up or invested in. A/B Testing is done in Azure by running the multiple versions of the applications side by side in so called 'slots' and splitting the traffic going to the application between the slots.

From this point on some assumptions will be made:

- One feature has been decided on to assess in this A/B Test
- A will be a control group using the standardized existing application, B will be a variation on A with a feature implemented
- The variables of success have been figured out beforehand
- The different versions of the applications are already developed

### Preparing the versions for A/B Testing

First, we will prepare the multiple versions of the application for A/B Testing. As mentioned before, A/B Testing is done in Azure by running the multiple versions of the application in so called 'slots'. The slots are in essence just deployment environments. However, to allow the collection of data from these slots a few additions need to be made to the versions of the application.

### Setting up Application Insights

An Application Insights workspace needs to be set up in Azure. This workspace allows for the collection and analysis of data from the different versions of the application. Setting up a workspace is simple, and this will not be covered in depth in this document. However, a guide on how to set up the workspace can be found [here](#).

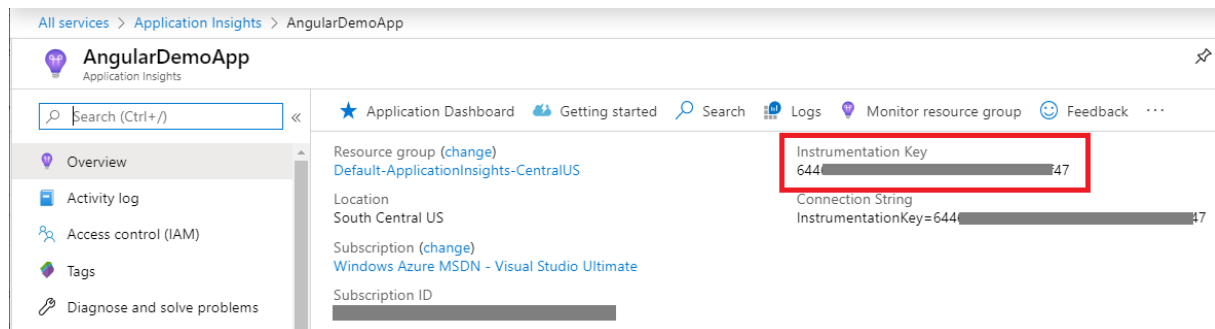
### Linking Application Insights to application

Up next, each version of the application needs a connection to the Application Insights. Right of the bat we want to install all needed NPM packages:

```
npm install --save @microsoft/applicationinsights-web
```

All versions of the application can use the same code snippet. The following bits of code need to be added to the Angular applications to work:

First, we want to define the instrumentation key in the Angular application. The best practice for doing so is defining this in the environment to keep it isolated from the code and allow for swapping keys if needed. Your instrumentation key can be found on the Azure platform on the Application Insights view:



### **Environment.ts**

```
export const environment = {
  ...
  appInsights: {
    instrumentationKey: '<your-guid>'
  }
};
```

Next, we want to be able to tag outgoing data with what slot the data comes from, resembling the A or B slot. When deploying a slot Azure automatically adds a cookie to each application that stands for the slot the application is running is. This cookie is called 'x-ms-routing-name'. We want to read this cookie and attach it to the calls made to Application Insight through a 'telemetry' class. Add the following file with corresponding code to the Angular project:

### **TelemetryInitializer.ts**

```
const telemetryInitializer = (envelope: any) => {
  const environment = getCookieValue('x-ms-routing-name') || 'production';
  envelope.data['slot'] = environment;
}

const getCookieValue = (key: string) : string | null => {
  const cookie = document.cookie
    .split('; ')
    .find(cookie => cookie.startsWith(key));

  return cookie ? cookie.split('=')[1] : null;
}
```

Furthermore, we need to add a bit of code that allows us to send the variables we want to monitor from our application to Application Insights. Add the following file with the corresponding code to the Angular project:

### **MonitoringService.ts**

```
import { ApplicationInsights } from '@microsoft/applicationinsights-web';
import { Injectable } from '@angular/core';
import { environment } from '../environments/environment';
import telemetryInitializer from '../telemetryInitializer';

@Injectable()
export class MonitoringService {
  appInsights: ApplicationInsights;
  constructor() {
    this.appInsights = new ApplicationInsights({
      config: {
        instrumentationKey: environment.appInsights.instrumentationKey,
      }
    });
    this.appInsights.addTelemetryInitializer(telemetryInitializer);
    this.appInsights.loadAppInsights();
  }

  logMetric(name: string, average: number, properties?: { [key: string]: any }) {
    this.appInsights.trackMetric({ name: name, average: average },
      properties);
  }
}
```

To make Dependency Injection work for the MonitoringService we need to add it as provider in our NgModule:

### **App.module.ts**

```
import { MonitoringService } from 'src/MonitoringService'; //Path Might
need to be changed

...
@NgModule({
  ...
  providers: [
    ...
    { provide: MonitoringService, useClass: MonitoringService }
  ]
  ...
})
...
```

Implementations need to be made to send data about the variables determined to check the effectiveness of the implemented feature. Monitorization can be added anywhere by using Dependency Injection to inject the MonitoringService and calling the logMetric method with the relevant data:

```
import { MonitoringService } from 'src/MonitoringService'; //Path Might
need to be changed
...
// Dependency inject MonitoringService in constructor
constructor(... private insightMonitor: MonitoringService ...) { ... }

// Call logMetric to send data to Application Insights from applicable
function

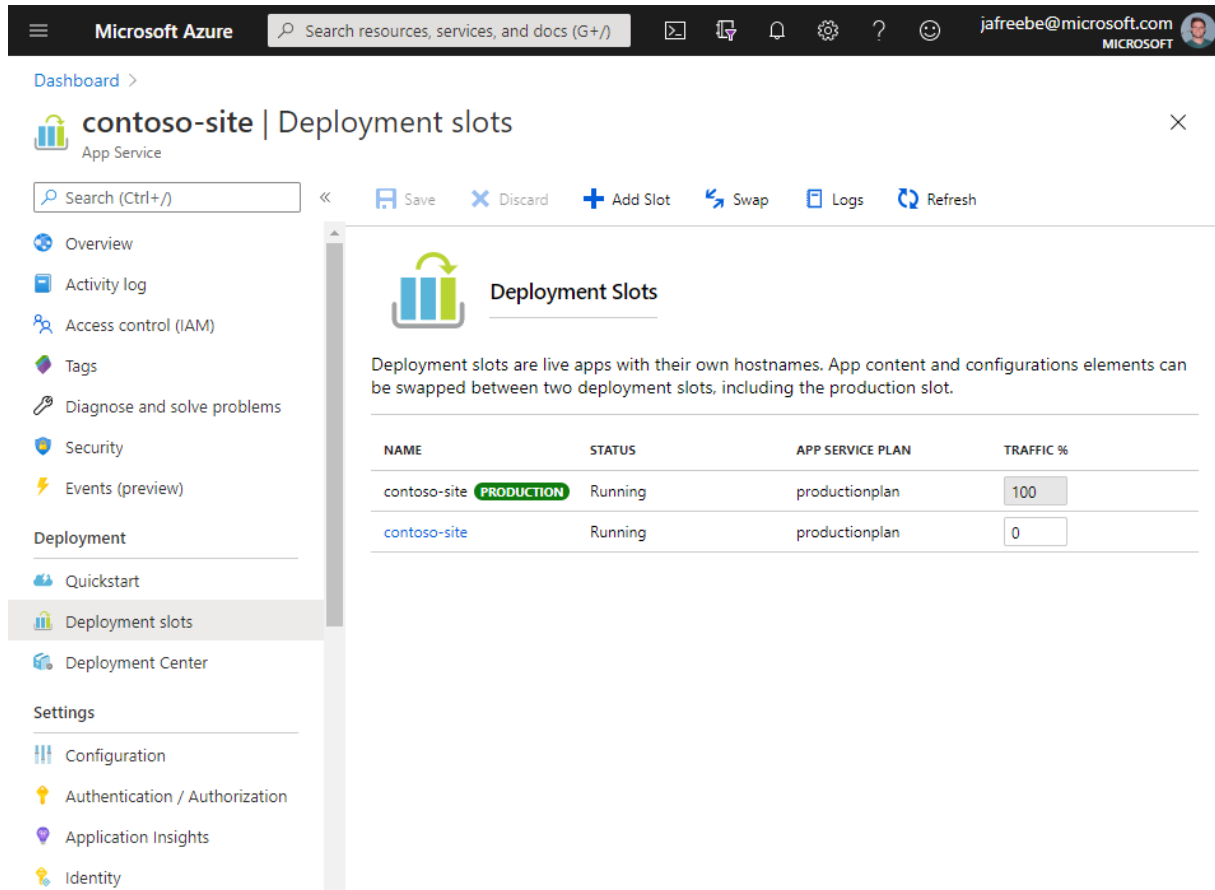
const foo = () => {
    this.bar = 20;
    this.insightMonitor.logMetric('bar', this.bar);
}
...
```



## Deploying versions

Finally, the different versions need to be deployed to Azure slots. CI/CD can optionally be used to automatically deploy to the Azure slots. If desired, a guide on how to set up the CI/CD for deploying to Azure slots can be found [here](#). If not desired, slots can manually be created for each version of the application. It is important to give each version you want to evaluate its own slot. In case of A/B Testing with two versions, two slots are needed.

Once the slots are created and the different versions of the application are running in their own slot, a desired routing-ratio can be configured on the Azure Deployment slots page by changing the **'TRAFFIC %'** value for each slot:



The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes the Microsoft Azure logo, a search bar, and user information. The left sidebar contains a navigation menu with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Events (preview), Deployment, Quickstart, Deployment slots, Deployment Center, Settings, Configuration, Authentication / Authorization, Application Insights, and Identity. The main content area is titled 'contoso-site | Deployment slots' and features a 'Deployment Slots' section. Below this, a table lists the deployment slots:

NAME	STATUS	APP SERVICE PLAN	TRAFFIC %
contoso-site <b>PRODUCTION</b>	Running	productionplan	100
contoso-site	Running	productionplan	0

With the slots running we now have all we need to run the A/B Test. Set your timeframe, let users use the application and gather data. All data will now, if set up correctly, automatically be collected in your Application Insights Workspace under Workspace > Monitoring > Metrics. This data can then be directly used to draw conclusions on the effectiveness of the implementation of a feature.

## References

- Anuraj. (2021, January 29). *A/B Testing with Azure App Service*. Retrieved from dotnetthoughts: <https://dotnetthoughts.net/ab-testing-azure-app-service/>
- Atkinson, L. (2020, March 29). *Angular How-to: Add Application Insights to an Angular SPA*. Retrieved from Microsoft Learn: <https://devblogs.microsoft.com/premier-developer/angular-how-to-add-application-insights-to-an-angular-spa/>
- Freeberg, J. (2020, August 3). *A/B Testing with App Service*. Retrieved from Azure Web Service: [https://azure.github.io/AppService/2020/08/03/ab\\_testing\\_app\\_service.html](https://azure.github.io/AppService/2020/08/03/ab_testing_app_service.html)
- Freeburg, J. (2020, June 29). *Zero to Hero with App Service, Part 2: Continuous Integration and Delivery*. Retrieved from Azure Github: [https://azure.github.io/AppService/2020/06/29/zero\\_to\\_hero\\_pt2.html](https://azure.github.io/AppService/2020/06/29/zero_to_hero_pt2.html)
- Microsoft. (2022, December 15). *What is A/B testing?* Retrieved from Learn Microsoft: <https://learn.microsoft.com/en-us/gaming/playfab/features/analytics/ab-testing/>
- Microsoft. (2023, March 14). *Application Insight Overview*. Retrieved from Learn Microsoft: <https://learn.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview?tabs=net>
- Microsoft. (2023, December 4). *Workspace-based Application Insights resources*. Retrieved from Microsoft Learn: <https://learn.microsoft.com/en-us/azure/azure-monitor/app/create-workspace-resource#create-a-workspace-based-resource>
- Optimizely. (n.d.). *What is A/B testing?* Retrieved from Optimizely: <https://www.optimizely.com/optimization-glossary/ab-testing/>