

Foize is a globally operating startup based in the Netherlands. It creates software solutions for digital social communication, geo-fencing and other location-based innovations.

Foize’s rapid growth has led to scaling challenges, particularly in their system’s GPS service, 24Coms. The team identified Infrastructure as Code (IaC), architecting for the Cloud and automation as keys to improving their ability to meet demand quickly. Throughout this journey, Foize aimed not only to improve their system, but to foster a “spirit of DevOps” within their team.

## Hacking Agility

Foize’s 24Coms solution is a hybrid solution, utilizing some Azure services and some on-premises services. The team realized quickly that their hybrid solution is not highly scalable. One of their biggest problems was deployment time for bug fixes. Deployment was a manual procedure that took between 2 and 4 hours to complete.

Re-architecting for the Cloud would take some time. In order to keep servicing their customer effectively, the team decided to initiate a “hackathon” – three days of focused effort to rapidly improve a few key areas. They didn’t focus on the entire process – just the areas that would have immediate value.

Don’t try to change everything at once. Focus on small changes that bring immediate value.

“

The team initially spent some time performing a Value Stream Mapping. After examining their value stream, they concluded that they could improve their Mean Time to Repair (MTTR) by focusing the “hackathon” on IaC, Continuous Integration (CI) and Continuous Deployment (CD), and automating Staging and Production environment creation.

## Using Visual Studio Team Services (VSTS) to Deliver CI

VSTS is a complete DevOps solution from Microsoft. It is itself a service hosted on Azure, Microsoft’s Cloud Platform. VSTS allows teams to manage source code, run automated builds with a cross-platform build engine, manage work items using Kanban boards and Agile Portfolio Management features, manage both manual and automated testing and manage releases.

The team quickly checked their source code into VSTS. They were then able to automated a build that runs every time a developer commits code to the repo (Continuous Integration). The next challenge was to deploy the code in an automated fashion.

## Continuous Delivery

The team then evolved the CI build to perform automated deployment to their staging environment, which is hosted in Azure. In order to perform the deployment, the team scripted the creation of the infrastructure that their solution depends on – Azure Service Bus Queues and Topics.

The team discovered that creating the Queues and Topics from code (IaC) was non trivial. The primary problem was that the Azure REST API is still in preview mode, and so some of the methods the team required were not available at the time. However, the team persevered and were able to work around the limitations because the VSTS builds agent can run any script. The team were able to use a client object model, invoked from a script, to work around the missing Azure REST methods. They now had automated their infrastructure creation.

Make sure your goal is measurable. Nothing helps drive DevOps adoption like seeing measureable improvements.

“

The team then used an out-of-the-box VSTS build task to deploy one of their services as an Azure Cloud Service. However, the Task seemed to be failing in unexpected ways. The team were able to download the source code for the Task from GitHub, where the Source Code for all the VSTS build tasks is publically available. They were able to analyze the failure, extract some code and create their own build step to work around the failure.

The team then worked on further augmenting the build to automatically deploy code as Worker Roles in Azure. The worker roles depend on existing Queues and Topics, so they get deployed after the IaC has created them. A few simple lines of PowerShell enabled the team to automate the worker role deployment easily.

## Measuring Success

The Foize team had already identified Mean Time to Repair (MTTR) as a measure they needed to improve on. The manual MTTR before the hackathon was between 2 and 4 hours on average.

After the hackathon, the MTTR was reduced to only 15 minutes – a 8 to 16x improvement! The team are understandably very happy with the results of their 3 days of intense focus.

Besides realizing immediate improvement, the team now have a blueprint they can reproduce to bring better DevOps to other applications and projects. Their success is inspiring them to keep improving and to keep changing for the better.

This case study is for informational purposes only.

MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.

Document published November 2015

*Ronald Koster (Android Dev, DBA), Leo Winder (Owner, Ops), Michael Vlaar (C# Dev, Ops)*