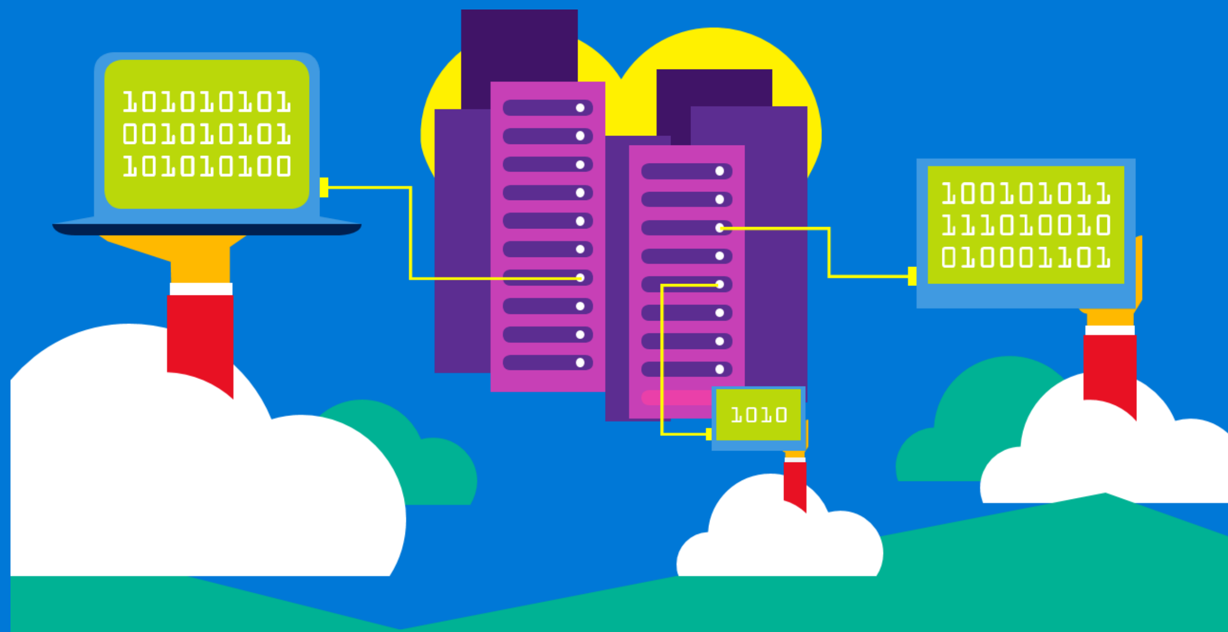
Microsoft



Example Azure Implementation for Government Agencies

Indirect tax-filing system

By Alok Jain
Azure Customer Advisory Team (AzureCAT)

June 2018

Contents

Introduction 3
Background 3
Background on the service providers 4
Technology requirements for the service provider application 5
Architecture 5
Azure technologies 6
Azure Active Directory 6
Web Apps 6
API Management 7
Azure ExpressRoute 8
DevOps process 9
Conclusion 10

List of figures

Figure 1. High level framework for the proposed tax submission process. 3
Figure 2. Architecture used by various service providers building applications. 5
Figure 3. Sample Azure ExpressRoute connection to the GTP datacenter using a dedicated private network. 8

Authored by Alok Jain. Edited by RoAnn Corbisier. Reviewed by AzureCAT.

© 2018 Microsoft Corporation. This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY. The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Introduction

Governments worldwide are looking for ways to enhance their ability to deliver services to their citizens. In this endeavor, they are adopting innovative methods that can not only enhance the service delivery capabilities, but also do it in the most cost effective, secure, highly available, and highly performant environment.

This paper describes an example architecture that can be used in such a project and explains how an architecture based on the Azure platform can help a government agency deliver tax-submission services to the end user.

Application functionality details are out of scope for this paper.

Background

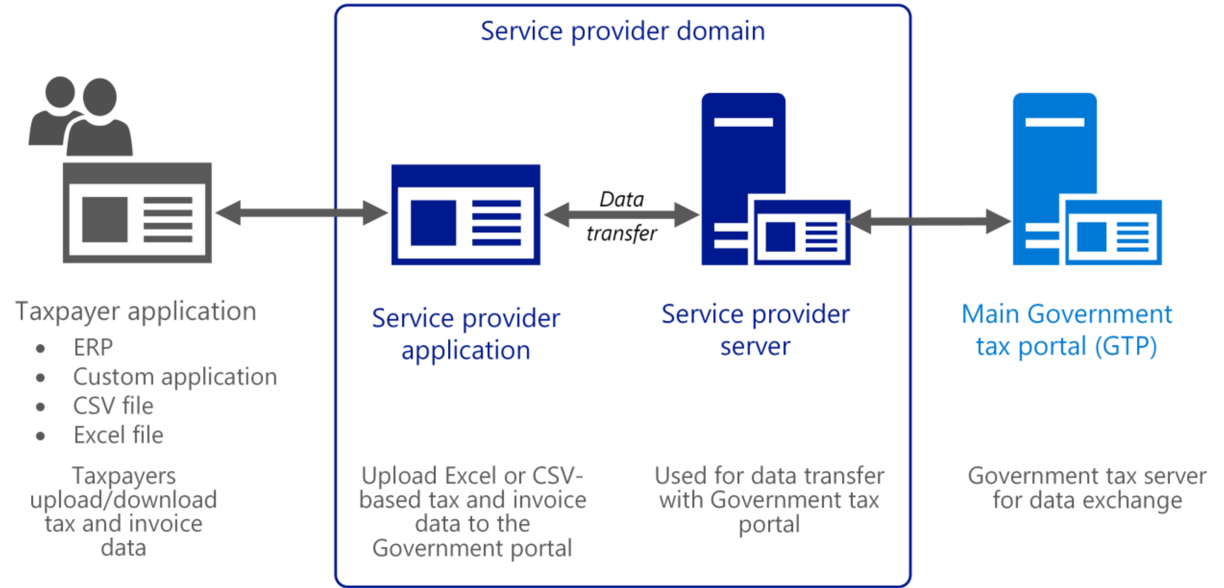


Figure 1. High-level framework for the proposed tax submission process.

The diagram above depicts a very high-level framework for an end-to-end corporate tax submission process, as defined by the government agency. The framework has been divided into three major parts.

The **first part, on the left**, depicts a tax payer application that is used by all big, medium, and small corporations to prepare the tax form, collate all the related tax information, hold the tax payer information, and finally for the submission of tax. This application and the hosted platform can be an in-house application developed by the corporation itself or as an application independently developed by an authorized application service provider. The number of such applications can run into the thousands as each corporation has the freedom to choose and develop the application that suits its own need. Although the UI used by an individual application can be different, each organization is expected to file the corporate tax in pre-defined formats at pre-defined intervals. At this time, each corporation files 12 different types of forms. Some of the forms are a one-time activity, such as registration, and some forms

are submitted on a monthly basis, such as filing monthly tax returns, filing for the refund of tax in case of an excess tax payment, and so on.

The **government main tax portal (GTP) system** is a G2B (government to business) portal used by corporate taxpayers to access the tax application. However, that is not the only way to interact with the GTP system. Instead, the taxpayer has their choice of third-party applications that will provide convenient user interfaces via desktop, mobile, and others, that will be able to interact with the GTP system.

The **middle part** of the diagram (titled "Service Provider Domain") represents the third-party applications or **service provider applications (SPA)**. The SPA acts as a layer in front of the GTP system and provides a convenient way for taxpayers and other stakeholders to interact with the GTP system, right from the registration of an entity to the uploading of invoice details, to the filing of returns. There will be two sets of interactions, one between the tax payer application users and the SPA and the second between the SPA and the GTP system.

This paper will go into detail about the architecture used for the SPA. The architecture in discussion is completely based on Azure and is being used by multiple service providers.

Background on the service providers

When a system is designed, the government can often feel that it is challenging for them to deal with the heavy load of monthly filing (such as 3 times a month) by all the corporations across the country. Tax filing on a monthly basis will not only put a heavy load on the system, but it will demand a huge investment of time and manpower to deal with the customer queries and complaints, especially during the peak load scenarios.

As a result, the government would appoint several service provider organizations to act as a frontend for the central government's tax portal (GTP). Approximately 34 independent service providers would be selected across the country. The profile of these service providers ranges from a big software development company, to a big financial company, to a big consulting company with a common thread of having rich domain experience in tax-filing applications. Major responsibilities of these service providers often includes, but is not limited to:

- Create a solution gateway that will allow end users to submit their tax forms to the portal, either via manually filing on the portal or via an app.
- Check the filed tax form for completeness, in terms of adhering to the latest tax rules.
- Submission of the tax forms to the government portal (connected via a dedicated private MPLS connection).
- Host an application that facilitates communication between end users and the government portal, in the case of a dispute on the forms submitted, or amounts paid.

Our team has worked with 15 service providers to help them build and host their applications on Azure. There were variations in the architecture based on the individual scenario and preference by the service provider (for example, a .NET-based application to Java-based development, SQL DB to NoSQL DB, and so on). Each of the service providers was an independent organization; hence they chose their own set of products and UI framework. However, the business functionality delivered by the architecture was the same.

Technology requirements for the service provider application

Before we go into detail about the architecture, let's look at some examples of high-level non-functional requirements for the SPA system, as would be outlined by a government:

- The SPA system will be hosted in a datacenter within the country.
- The SPA system should be deployed in a highly-available, multi-region configuration within the datacenter to provide high availability even in case of datacenter failure.
- The SPA system will provide a secure registration experience for the taxpayer.
- The SPA system needs to expose all the functionality via APIs in RESTful, JSON-based, and stateless services formats.
- The SPA system will interact with the government's GTP system via RESTful APIs over an MPLS-based private network, which connects the SPA to the GTP datacenter. This is to ensure controlled access of APIs and to avoid a single point of failure.

Architecture

After a detailed analysis of the requirements and considering various possible options (such as IaaS versus PaaS, API gateway versus direct API integration, and RDBMS versus NoSQL DB), the following architectural framework was designed.

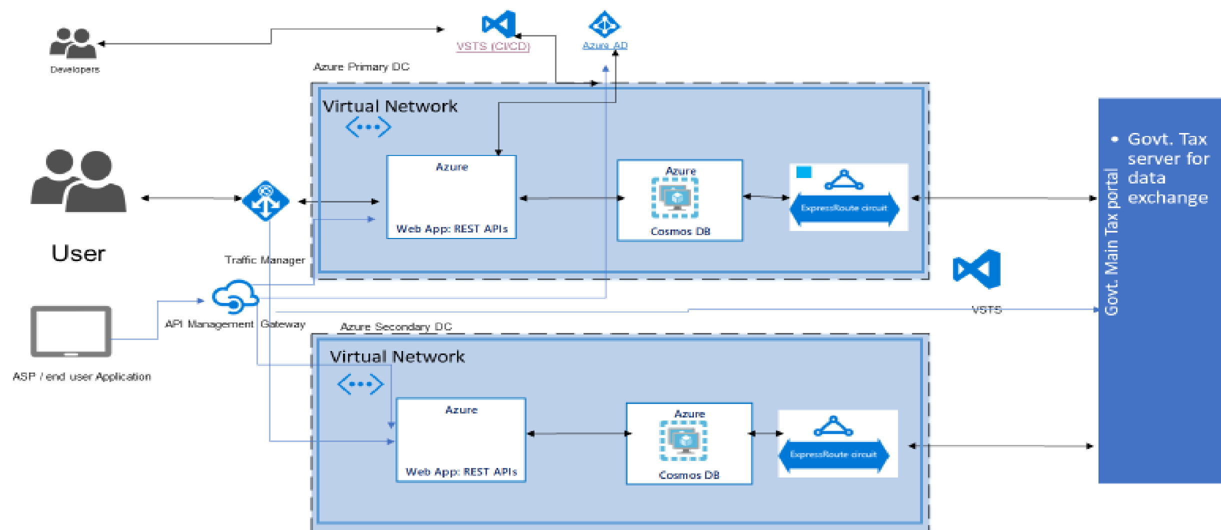


Figure 2. Architecture used by various service providers who are building applications.

The architecture has the following salient points:

- The architecture is based on PaaS components, which offload many of the maintenance and management tasks.

- All the technologies are deployed in a highly available configuration, which ensures the availability of the solution at all times.
- NoSQL DB was used to store the tax data. This provides the flexibility of having a dynamic schema.
- The Web App service is used to host the SPA application, and it provides the flexibility to host either Java or .NET-based applications in the PaaS environment.
- An API gateway is used for the API calls between the taxpayer's application and SPA, and between the SPA and GTP application.
- Security is addressed at multiple points, such as when storing data in encrypted form, authenticating and authorizing the user, deployment in secure virtual network configurations, and so on.
- The architecture provides the flexibility of storing the user's data, either in Azure Active Directory or NOSQL DB.

Azure technologies

Next, we'll discuss the Azure technologies used in the multi-region deployment of the architecture, and we'll also suggest pointers for the deployment of the various services.

One of the key requirements for the solution was to deploy it in a highly-available, multi-region configuration within the datacenter, which provides higher availability to protect it from instance failure, virtual machine failure, and even from a region failure. The architecture (figure 2) was deployed in an **active/passive** topology with hot standby, using Traffic Manager for failover.

For a detailed explanation of how to achieve a highly-available multi-region deployment, along with step-by-step instructions for the deployment of the solution, see Multi-region deployment. It also provides details about the various scenarios for the multi-region deployment and covers the deployment of some important components, like Traffic Manager, SQL Database, CosmosDB, and Storage.

The following services are used in the architecture.

Azure Active Directory

Azure Active Directory (Azure AD) is used for storing user's identity, and allows them to sign in securely. As part of this project, all the user credentials and authorization rights were stored in Azure AD. If you are unfamiliar with basic concepts of authentication in Azure AD then you can refer to Authentication scenarios for Azure AD. For information on how to add new users and set up the authorization rights for the user, see Quickstart: Add new users to Azure Active Directory.

Web Apps

The Web Apps feature of Azure App Service is used to deploy SPA. Web Apps is PaaS-based, it provides an environment to host your application, and it's deployed using Java, .NET, and other environments, such as PHP, Node.js, and so on. The primary environments that were used to develop SPA were either Java or .NET.

If you are developing your application in a Java environment, see Java application deployment to get step-by-step instructions. Similarly, if you plan to develop your application in a .NET environment, then see .NET application deployment for instructions. To configure your web app to use Azure AD for authentication, see Configure your App Service App to use Azure Active Directory login.

To learn about deploying an application in a multi-region configuration, see the Multi-region deployment reference architecture.

API Management

All the communication between the taxpayer's application and an SPA, and between an SPA and the GTP, is expected to be via REST API calls. Therefore API Management is a great option.

Azure provides the API Management service to publish and manage a web API. Using this facility, an SPA can generate a service that acts as a façade for one or more web APIs. The communication between the taxpayer application to an SPA , and an SPA to the GTP, will happen via API Management, which also known as a gateway.

API Management is a scalable web service that an SPA can create and configure by using the Azure portal. You can use this service to publish and manage a web API, as follows:

1. Deploy the web API to a website, Azure cloud service, or Azure virtual machine.
2. Connect the API Management service to the web API. Requests sent to the URL of the management API are mapped to URLs in the web API. The same API Management service can route requests to more than one web API. This enables aggregation of multiple web APIs into a single management service.
3. For each web API, specify the HTTP operations that the web API exposes together with any optional parameters that an operation can take as input. It can also be configured to see whether the API Management service should cache the response received from the web API, which optimizes repeated requests for the same data.
4. SPA can either define operations manually using the wizards provided by the Azure portal, or you can import them from a file containing the definitions in WADL or Swagger format.
5. Configure the security settings for communications between the API Management service and the web server that is hosting the web API. The API Management service currently supports basic authentication and mutual authentication using certificates and OAuth 2.0 user authorization.
6. Create a product. A product is the unit of publication; you add the web APIs (that you previously connected to the management service) to the product. When the product is published, the web APIs become available to developers.
7. Configure policies for each web API. Policies govern aspects, such as whether cross-domain calls should be allowed, how to authenticate clients, whether to convert between XML and JSON data formats transparently, whether to restrict calls from a given IP range, usage quotas, and whether to limit the call rate. Policies can be applied globally across the entire product, for a single web API in a product, or for individual operations in a web API.

For more information, see the API Management documentation.

Azure ExpressRoute

One of the requirements for the architecture is to have a dedicated MPLS-based network connectivity between the SPA datacenter and the GTP datacenter. The dedicated connectivity is required from both the primary data center as well as the secondary data center.

In our example architecture, [Azure ExpressRoute] is used to address how to securely and privately connect the SPA application hosted on an Azure datacenter to the GTP datacenter. The ExpressRoute connections use a private, dedicated connection through a third-party connectivity provider (generally an in-country telecommunication service provider).

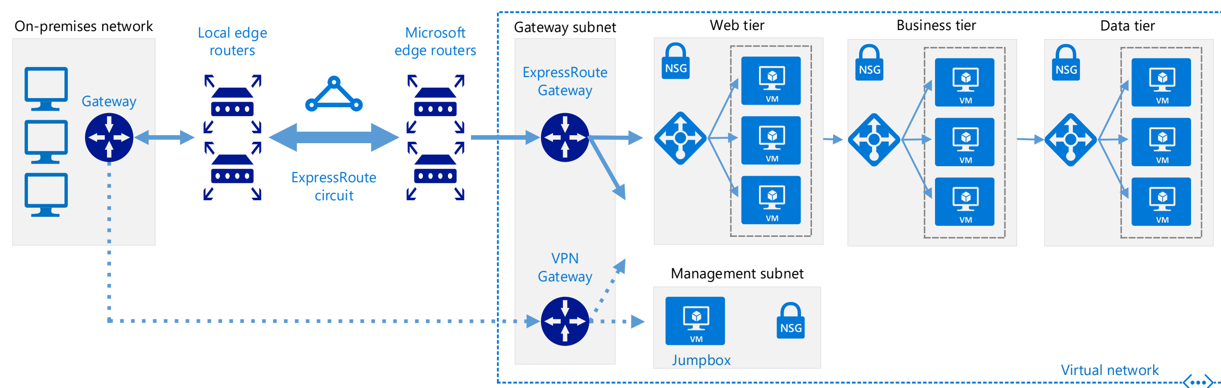


Figure 3. Sample Azure ExpressRoute connection to the GTP datacenter, using a dedicated private network.

The above figure shows a **sample configuration** in which Azure ExpressRoute is configured to connect with the on-premises network (in our case the GTP datacenter) over a dedicated private network. You can refer to [ExpressRoute reference architecture] for more information.

Some of the major components used for the ExpressRoute connectivity are as follows:

- **On-premises corporate network (in our case the GTP network)**. A private local-area network running within the GTP datacenter.
- **ExpressRoute circuit**. A layer 2 or layer 3 circuit supplied by the connectivity provider that joins the on-premises network with Azure through the edge routers. The circuit uses the hardware infrastructure that is managed by the connectivity provider.
- **Local edge routers**. Routers that connect the on-premises network to the circuit managed by the provider. Depending on how your connection is provisioned, you might need to provide the public IP addresses used by the routers.
- **Microsoft edge routers**. Two routers in an active-active highly available configuration. These routers enable a connectivity provider to connect their circuits directly to their datacenter. Depending on how your connection is provisioned, you might need to provide the public IP addresses used by the routers.
- **Azure virtual networks**. Each virtual network resides in a single Azure region, and it can host multiple application tiers. Application tiers can be segmented by using subnets in each virtual

network. The Web Apps environment for an SPA was configured inside a virtual network and was accessible only on a private IP address.

- **Azure services**. Azure compute services, namely Virtual Machines (IaaS) and Cloud Services (PaaS), and in our case a web app, was deployed within a virtual network connected through the private peering domain. The private peering domain is a trusted extension of the on-premises core network and provides the secure reliable connectivity between the GTP datacenter and Azure. It was set up for bi-directional connectivity between the GTP network and Azure virtual networks. This peering connected to the Web Apps service directly on their private IP addresses.
- **Connectivity providers** (not shown). Companies that provide a connection either using layer 2 or layer 3 connectivity between your datacenter and an Azure datacenter.

One of the major requirements for the project was that the traffic between the SPA datacenter (in this case Azure datacenter) and the GTP datacenter should be encrypted, that is the traffic flowing over an ExpressRoute circuit should be in encrypted form.

Currently, ExpressRoute circuits do not support data encryption by default. However, with the help of router service providers such as Cisco, Juniper, and so on the data can be encrypted over the ExpressRoute circuit. The configuration needed for the setup depended upon the network provider and router differing from vendor to vendor.

As an example, a Cisco ASR 100 router can be used. For instructions on how to setup a Cisco ASR 100 router with an ExpressRoute circuit, see Cisco ASR1000 and Microsoft Azure ExpressRoute Joint Validated Design . The Cisco router is an example only, and users are free to choose any other router and service provider. The ExpressRoute configuration details can be obtained from the router and network service provider you select.

DevOps process

The proposed architecture framework suggests a Web Apps environment that provides the flexibility of deploying the applications that use either a .NET environment or Java as the primary development language. In our experience, we found that most of the service providers developed their applications using open source technologies and used Java as the primary language for the development.

The Web Apps environment supports the deployment of Java applications from a development environment, by using various methods such as a GitHub repo or OneDrive, using Visual Studio Team Services (VSTS).

For simplicity, we recommend the use of trusted VSTS DevOps process for the deployment of the application. If you're a Java developer who's new to VSTS, see the Java introduction for useful tips. For a step-by-step approach for using VSTS to configure a CI/CD pipeline for Azure Web Apps, see Build and deploy to an Azure Web App.

To streamline the deployment cycle for the application we used slot swapping. Slot swapping facilitates deploying an application in a separate live App Service environment, and when it is ready to be used for production, one can swap the slots from staging to production. This provides seamless traffic redirection, minimizing disruption or downtime. For instructions on setting up automated slot swaps, see Azure DevOps: automate App Service slot swaps in your VSTS release pipeline.

Conclusion

The example architecture described here is a result of our experience in working with multiple partners who were selected as service providers by a government agency. The architecture provided a practical approach for building a solution that fulfilled the government's requirements for the tax application.

The Azure-based architecture was created strictly in accordance to the guidelines laid down by the government. We understood that there could be many variations in building the architecture, as some of the partners preferred to go with the IaaS environment rather than the complete PaaS environment, and some preferred a mix of IaaS and PaaS environments.

Azure provides flexibility to build architectures by using any of the variations, such as complete IaaS, complete PaaS, or mix of IaaS and PaaS. We therefore changed our approach, considering the preferences of the individual partner. In the end, we ended up with 5 partners that went live with a complete PaaS solution, 4 partners used a mix of PaaS, IaaS, and the rest used the complete IaaS-based solution. Almost all the partners also used third-party products as well. We conducted a workshop for each partner to finalize their architecture, and help was extended for multiple partners during their go-live period.

It was an enriching experience to help the partners build and deploy a robust, scalable, and secure solution, using Azure as the cloud platform. We also ended up learning a lot on the journey.