Azure DevOps Pipelines: Discovering the Ideal Service Connection Strategy

Introduction

When an organization is trying to configure their Azure DevOps (ADO) environment to deploy into Azure, they are immediately met with the dilemma on how their DevOps instance will execute the deployment against their Azure Environment. This article will go over the various configurations, decisions, and pros and cons that should be evaluated when deciding how your DevOps environment will deploy code into Azure.

This article will not talk about the nitty gritty details on how to configure the connection. This is covered in MS documentation. Nor will we discuss which type of authentication should be created. There are additional resources that will cover this. This article instead will focus on questions such as "How many Service Connections should I have?", "What access should my Service Connection have?", "Which pipelines can access my Service Connection?", etc...

Deployment Scope

This question on how to architect our Service Connections, the means by which Azure DevOps communicates to Azure, will be the main focal point of this piece. Deployment Scope, for the purposes of this article, will refer to what Azure Environment and resources our Azure DevOps Service Connection can interact with.

This answer will vary depending on your organization's security posture, scale, and maturity. The most secure will be the smallest deployment scope and will then entail the most amount of overhead, while on the flip side the least secure will have the largest deployment scope and least amount of overhead associated with it. We will cover three scenarios and the associated pros and cons: One Service Connection to Rule Them All, a Service Connection per Resource Group, as well as a Service Connection Per Environment.

As for what access the identity from ADO should have in Azure I typically recommend starting with Contributor as this will provide the ability to create Azure resources and interact with the Azure Management Plane. If your organization in leveraging Infrastructure as Code (IaC) I would also recommend leveraging User Access Administrator, to provision Role Based Access Controls and allow Azure to Azure resource communication leveraging Managed Identities. This is effectively the same permission combo as Owner; however, if you are familiar with Azure recommended practices, Owner permission assignment is not recommended in the majority of cases.

One Service Connection to Rule Them All

This scenario is pretty self-explanatory. A team would have a single Service Connection which all their pipeline would leverage across all environments. This scenario could be scoped a little more specific where a

specific application will have one Service Connection that is responsible for deploying across all environments.

Pros

The pro to this approach is pretty simple. We are only concerned about one Service Connection for our application(s). This means we have to manage and provision access to one identity behind the scenes in Entra ID. Additionally, if your organization is leveraging a Service Principle and password to authenticate to Azure these credentials will expire and need to be rotated. Since there is one Service Connection for all deployments, we will only need to maintain the connection to one Service Principal. This negates the potential risk your production deployment is delayed due to credentials expiring. This overall would be considered a small concern, unless it is realized while in the middle of a production outage/hot fix.

Cons

As mentioned earlier this would be the least secure approach. By enabling one Service Connection access to everything we are effectively given every deployment regardless of the deployment scope. This also means from a deployment/audit log that all activity will report back to one identity which can make troubleshooting and tracking harder.

We also introduce the unintended risk of a deployment designed to go a development environment inadvertently going to production. Defining the Service Connection is one criteria to separate Azure Deployments across environments. This is a rather large concern as someone can easily unknowingly forget to update the service connection and deploy code in the wrong location.

Service Connection Per Resource Group

The opposite of a one Service Connection to Rule Them All would be a Service Connection Pre Resource Group. This structure would have the finest grain access control where each individual Resource Group would have a Service Connection tied to it's deployment.

Pros

This approach is the most secure. It is the most restrictive model that is easily available when creating the Service Connection. By creating a Service Connection per Resource Group we are limiting what specific deployments can interact with. This would eliminate the risk of Team A's deployments touching Team B's resources since the deployment scope of the Service Connection would be limited to the resources contained in Team A's Resource Group. This can be a rather significant gain; however, I will point out that a proper CI/CD flow should mitigate this risk.

Additionally, from a logging and auditing perspective there would be clear transparency on what deployment updated what resources. Since each deployment pipeline's permissions are scoped to that pipeline, we can quickly trace back any changes that were made back to the orchestrating pipeline.

Cons

By the nature of this approach there will be exponentially more Service Connections to manage. If wanting to use this approach I would highly encourage using the Workload Identity Federation when setting up your connections. This will help alleviate the password expiration of your Service Connections.

If you aren't using Workload Identity Federation this could lead to the team having to constantly watch for and update passwords since a large organization could easily have 100 Resource Groups which means typically 100 pipelines and that could be across three environments. Thus, we are talking about managing and organizing around 300 Service Connections. Not a simple task.

Another significant drawback to this approach occurs when you are deploying Infrastructure as Code or having to update resources that live outside of the designated Resource Group. Since the Service Connection can only interact with the resources in its designated Resource Group it will not be able to provision any additional RBAC access required to resources that live outside of the Resource Group. Again, the scope of this effort in the example above could mean tracking access for 300 Service Connections.

Some common scenarios for this would be activities such as being able to setup up access to retrieve secrets from a shared Key Vault, add a Managed Identity access to a share data source, and pushing images to a shared container registry. To accommodate for this one would have to manually add the extra permissions. This then will have an impact as this access provisioning would not be defined via IaC and runs the risk of being unaccounted for in an organization Azure deployment.

A Service Connection Per Environment

This configuration is a bit of a goldilocks between the two extremes. In this scenario our Service Connection will have access to all resource groups in a specific Azure Subscription/Environment. The Service Connection would be reused across a team(s) to handle Azure Deployments to a specified subscription.

Pros

This architecture will lead to higher developer enablement as developers would be able to, via Azure DevOps Deployments, make any and all changes required for their applications and/or infrastructure. Additionally, by having different Service Connections we create the need to have the Service Connection updated when moving across environments. This is a significant win as it will satisfy any audit/security requirements that environments are separated by access.

This approach limits the activity of the Service Connection's role assignment to just one per environment as opposed to having to continuously create/update Service Connection role assignments each time a new pipeline is created. This is sizable as it will negate the overhead involved with maintaining these.

Cons

Structuring your Service Connection this way will violate the least privilege access principle. This principle is defined as:
"The information security principle of least privilege asserts that users and applications should be granted access only to the data and operations they require to perform their jobs."

Since the deployment pipeline will have access greater than what is needed to do its specific job (i.e. access to update resources outside the scope of the defined deployment) this architecture will violate this principle. As such this approach is not the most secure. In some organizations that strictly adhere to the concept of least privilege this could be significant and a deal breaker.

Auditing your Azure Environment will also not be as granular/transparent as it would be under the Service Connection Per Resource Group model since the same Service Connection would be reused across the environment.

Conclusion

These three approaches are all options when looking to configure your Service Connections from Azure DevOps into Azure. Each approach will have its one set of pros and cons your organization should evaluate when making a decision. It is important to understand there isn't one universal answer for everyone.