CI / CD

Background & Purpose

The situation at Contica right now is that our various delivery teams do not follow a standardized way of handling and deploying the source code for our solutions. Our main focus has been the ability to deploy all of our resources and remove as much manual deployment and configuration steps as possible. Thus, we have built a portfolio of deployment scripts and templates while simultaneously increasing our individual and collective knowledge. It's no longer a question about if we can deploy something, but how.

Now that we have acquired a lot of knowledge and experience within the subject, it is time to focus on a concise way of working with the CI/CD process across teams. In this document, we have put forward our stance within this subject with the resources and collective knowledge available at Contica today.

Purpose of this initiative

- To represent our stance on how we can standardize parts of our CI/CD process and work more uniformly across delivery teams and customers.

Not the purpose of this initiative

- To be a general best practice guide for the standardization of a CI/CD process.
- To be a finalized product immune to changes and improvements.

In this document, we have answered the following questions:

1. In what technical way do we separate our source control and deployments?
2. How do we determine what is code and what is infrastructure?
3. What do we currently view as code and infrastructure in Azure?
4. When should we use ARM templates vs PowerShell Scripts, and do we have a preference?
5. When should we share templates and scripts across repositories?

Analysis & Conclusions

In what technical way do we separate our source control and deployments?

Today we have implemented CI/CD in one of two ways:

1. One repository and pipeline for the whole integration.
    - Takes a longer time, possibly unnecessary deployment of unchanged resources, slower deployment.
    - Fewer repositories, all source code represents the state of the whole solution/integration.
2. One repository with separate branches for infrastructure and code with a shared pipeline.
    - Separate branches that will never interact with each other, never be merged.
    - No way of separating access to infrastructure and code.
    - Parts of the same code-solutions (e.g., logic app standard) per definition may belong to different categories, which may complicate the implementation of the infrastructure and code separation.
    - Less unnecessary deployment of unchanged resources, faster deployment.
    - Less repositories.
    - More precise and quicker disaster recovery.

At the current stage, we have concluded that the best standard for us to practice is by:

- Separate repositories and pipelines for code and infrastructure.
- The possibility of separating access to infrastructure and code (security concern).
- Less unnecessary deployment of unchanged resources, faster deployment.
- More precise and quicker disaster recovery.
- Clear separation of the resources and related changes.
- More repositories.
- More pipelines.
- Parts of the same code-solutions (e.g., logic app standard) per definition may belong to different categories, which may complicate the implementation of the infrastructure and code separation.

Additional notes:

- Apis & operations source code should be maintained and deployed separately from the related integrations and consumers, as an API may be consumed by or related to several integrations. By deploying and maintaining the source code per API, we'll also make sure that we perform fewer unnecessary deployments of unchanged APIs.
- Role assignments should be handled per integration in a separate repository and pipeline, mainly for security concerns.
- Since the deployment of role assignments requires a service connection with a User Access Administrator or Owner role, it is of utmost importance that additional security measures are applied regarding the access to and usage of such service connections.
- To ensure that the development and deployment of integration resources is kept seamless while simultaneously providing tight security measures and protection of the Azure subscription, a separate duo of service connections should be used for each Azure subscription: one service connection with the right to deploy role assignments and one service connection to deploy Azure resources.

In conclusion, we will be using separate repositories for:

- Code: One repository and pipeline per resource group.
- Infrastructure: One repository and pipeline per resource group.
- APIs: One repository and pipeline per API.
- Roles: One repository and pipeline per set of roles for an integration.

How do we determine what is code and what is infrastructure?

"Code refers to the computer program algorithms, made up of symbols from a source alphabet, that represent the set of rules on what actions the program is expected to perform."

"Information technology (IT) infrastructure are the components required to operate and manage enterprise IT environments. IT infrastructure can be deployed within a cloud computing system or within an organization's facilities."

To be able to know what category each Azure resource belongs to, it's important to know the definition of each category and where they differ.

When we speak about code, we refer to executable code and any instructions, transformations, or logic. Whereas the infrastructure is the components required to operate and manage enterprise IT environments and support the execution of code.

The main question one should ask themselves is: What is the function of the resource?

- If it is executable instructions for the transformation of data or other types of logic, then the resource is classified as code.
- If it is required to operate and manage the IT environments or support the functionality of code, then the resource is classified as infrastructure.

In addition to determining if the resource should be classified as code or infrastructure, one should always consider possible security risks and how the deployment of the new resource affects the current deployment process, depending on where and how the source control and deployment are handled.

What do we currently view as code and infrastructure in Azure?

Code in Azure

- Logic app consumption
- Azure Function
- Workflows
- Apis & Operations
- Maps and transformations (e.g., XSLT, Liquid)
- Logic app standard
- Azure Function App

Infrastructure in Azure

- Service bus namespace and its related resources
- Storage Account & child resources
- Resource Group
- API Management
- Virtual networks & child resources
- Key Vault
- Integration account
- Virtual Machines
- Databases
- SQL Servers
- Event Hub
- Gateways
- API Connections
- Log analytics

- Application insights
- Diagnostic Settings

When should we use ARM templates vs PowerShell Scripts, and do we have a preference?

We suggest ARM templates over PowerShell wherever possible:

- ARM templates describe the result, not the process.
- ARM templates are not as sensitive to being outdated, requiring less regular maintenance.
- Having only ARM templates would make us less dependent on our current deployment environment and tools (less work in changing to GitHub or others for deployments).

When should we share templates and scripts across repositories?

Use a combination of both:

- **Utils repository**: Shared general templates and scripts for resources that differ less between integrations. Easier to manage and maintain (updates to API version, etc.).
- **Integration/solution repository**: More specific templates and scripts more suitable for a specific case. Using what's best for the scenario. Not as much need for replacing existing templates in utils repository and potentially losing valuable work. Utils repository will differ less across customers.

To further reduce the risks for our shared utils repository, we should start applying our branching strategy for our shared scripts and templates.

What could be next?

Notes for the reader: How can we build upon this base, things like further automating, standardization of rules and policies, etc., in order to bring us closer to defining our CI/CD process at Contica.