TEMPLATE-BASED WEB APP AND DEPLOYMENT PIPELINE ON AZURE IN AN ENTERPRISE ENVIRONMENT

Dr. Johannes Schöck
Senior Expert Data Science, NKD Group GmbH

PyData Berlin
02.09.2025

ABOUT ME // CODE AND SLIDES

Johannes Schöck

- Studied molecular nano science, PhD in physics
- Self-taught coding and data science skills
- Transitioned into data science since 2019
- Data Scientist at NKD since 2023

Find me on LinkedIn

Code and slides @GitHub

https://github.com/JSchoeck/talk_webapp_template_and_pipeline_on_azure


NKD
NKD
NKD
NKD
NKD
NKD
30%
30%
Große Auswahl an Heimtextilien,
Wohnaccessoires und Haushaltswaren
1€
7€
9€
15€

Key Facts and Figures

Leading platform in the attractive **discount sector**

~2,200 profitable, smaller format stores across 7 countries

+10,000 employees

820 Mio turnover in 2023

3.3% sales growth since 2018

Our Corporate Mission

NKD - A good feeling every day - for our customers, for our employees, for society. As a sympathetic discounter, we act responsibly, are always close by, and offer everyone an attractive value-for-money ratio.

European Footprint

Category Overview

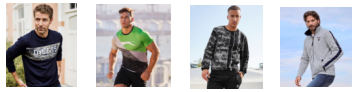
Apparel, +75%

Home, 15-20%

Other, 5-10%

Womenswear (64% of Apparel)

Menswear (19%)



Kidswear (17%)

Homeware

Other

NKD GROUP

NKD Online Shop
https://www.nkd.de/

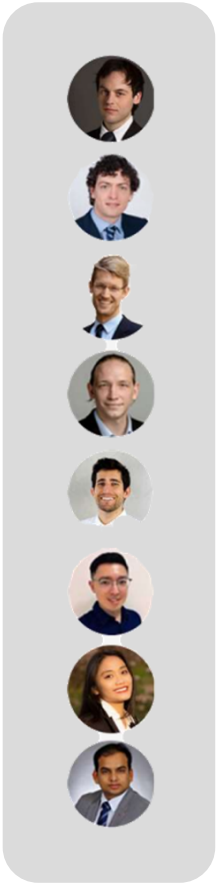
NKD Career
https://www.nkdgroup.com/

NKD Data Science Team

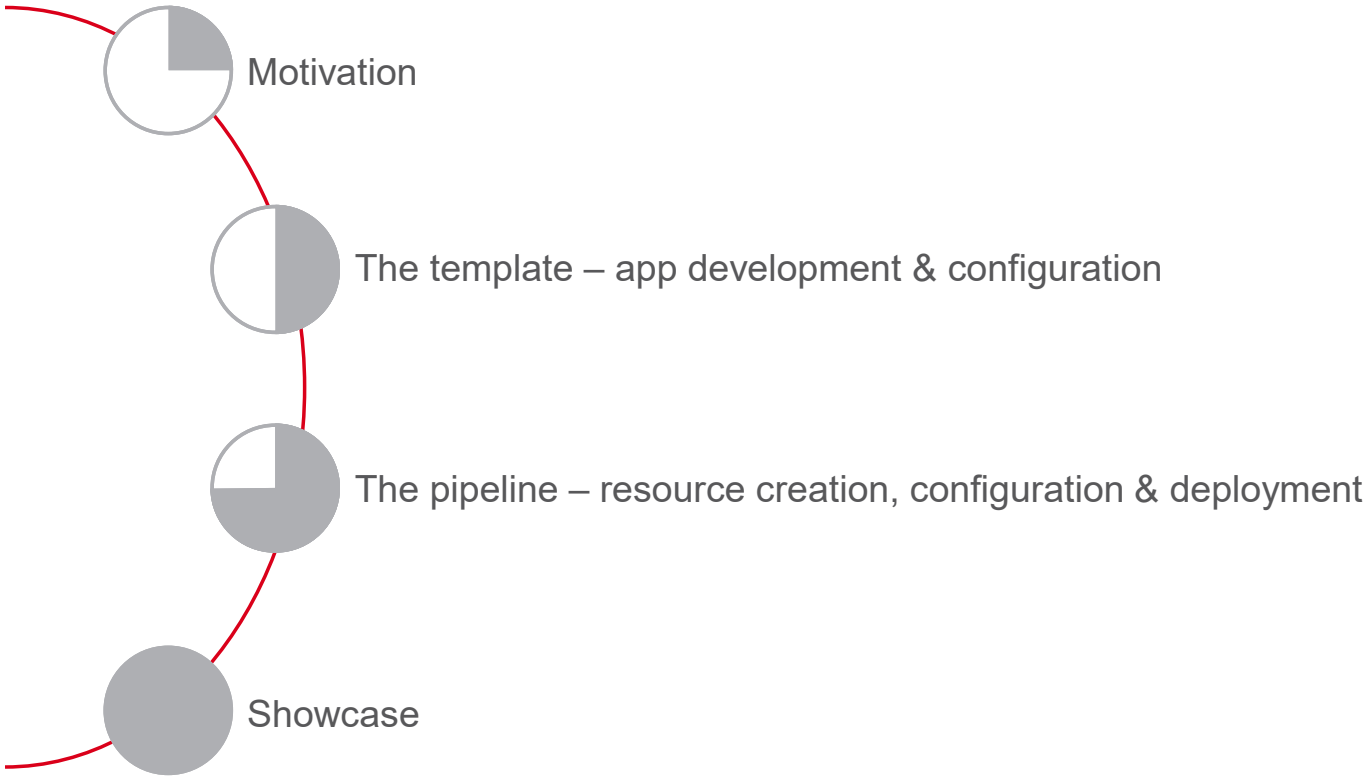
- 6 Data Scientists
- 1 Cloud Engineer
- 1 Working Student

\+ MIS Team for data and database topics

Pipeline and template co-developed with Luis Cuervo & Sai-Ram Nallamothu



OUTLINE


Motivation
The template – app development & configuration
The pipeline – resource creation, configuration & deployment
Showcase

OUTLINE

1. Motivation
- Why it's hard to get user feedback early and why that is problematic
- Why it's hard to get a real application running early
- What if we could automate app deployment and configuration, or how the NKD data science teams went from awkward to awesome

2. The app creation, deployment, and configuration process
- Struggles and best practices
- Tools that help with consistency and automation
- Handling virtual environments across dev systems and the cloud
- Web app and pipeline repositories and templates

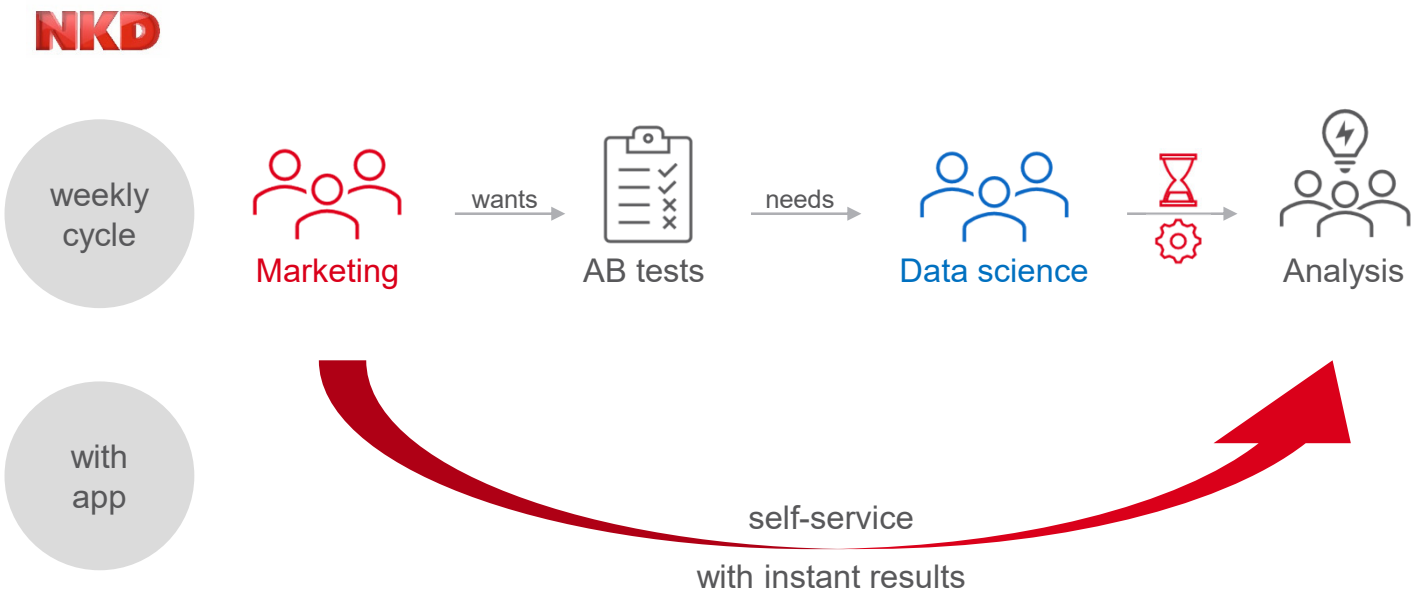
3. The pipeline
- Structure of the stages
- Minimizing manual configuration with file parsing and bicep
- Matching branch and target server
- Automated Azure resource creation using Azure CLI
- App authorization and authentication configuration with more Azure CLI
- Finally, the deployment

4. Show case
- What the setup looks like when it's fully set up
- Deploying an app live

NKD

MOTIVATION

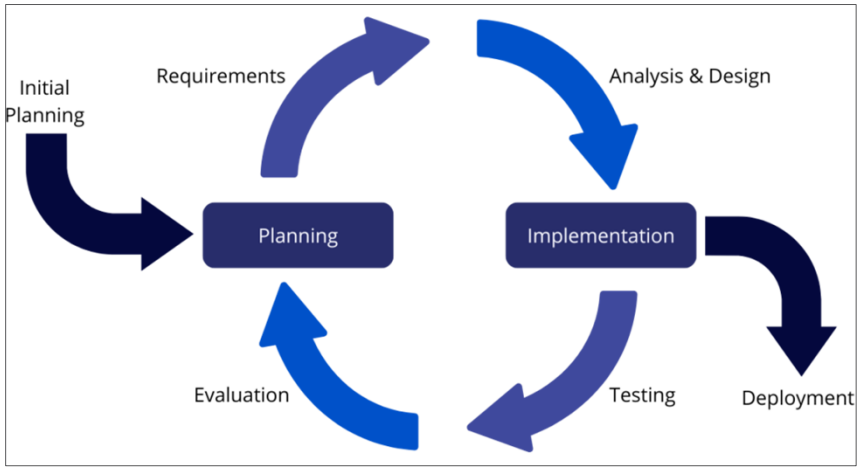
- To increase the **impact of data science teams**, they should not work on user requests
- Instead, companies should **enable business users** to get the data and predictions they need for a specific task on their own
- Building **data-driven applications** empowers business users to be faster, more informed, and make better decisions



MOTIVATION

Developing a data-driven app – requirements

- Solving the problem at the core of the task
- Using the right data
- Understanding the data



It's hard to get user feedback early

- Complex data interactions: filter, sort, aggregate, time shift…
- Nothing to show to the business users during backend development
- Business users often have limited knowledge about data-driven app peculiarities

…and that's problematic!

- Business rules impact data, software and UI design decisions, but often are not communicated up front
- No feedback loop for iterative improvement – worst case: building the wrong thing
- Changing requirements late in the process increases development time and code complexity
- Poor user experience
- Lack of trust
- Low adoption rate

Image source: https://www.datprof.com/

MOTIVATION

Why it's hard to get a real application running early

- With business users, discussing an app based on its code is usually not possible
  → UI required

- Setting up an app locally limits how business users can interact with it
  → lowers buy-in, does not scale later

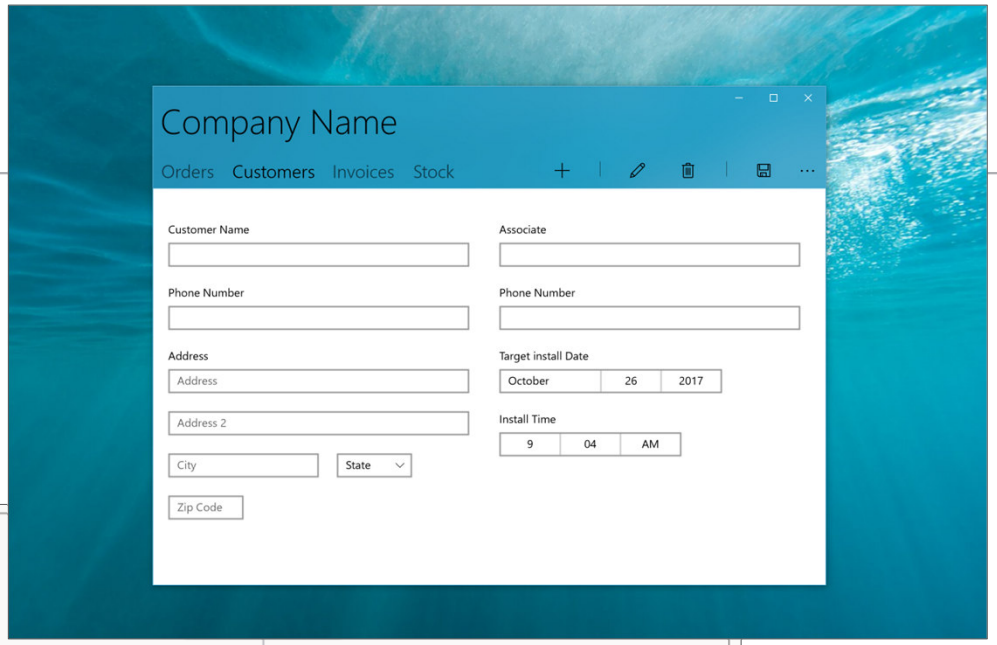
- Deploying an app in the cloud leads to additional issues
  - Dependencies
  - Interaction of data, backend, and frontend
  - Budget approval / costs
  - Knowledge, effort & permissions to setup infrastructure
  - User authentication & authorization

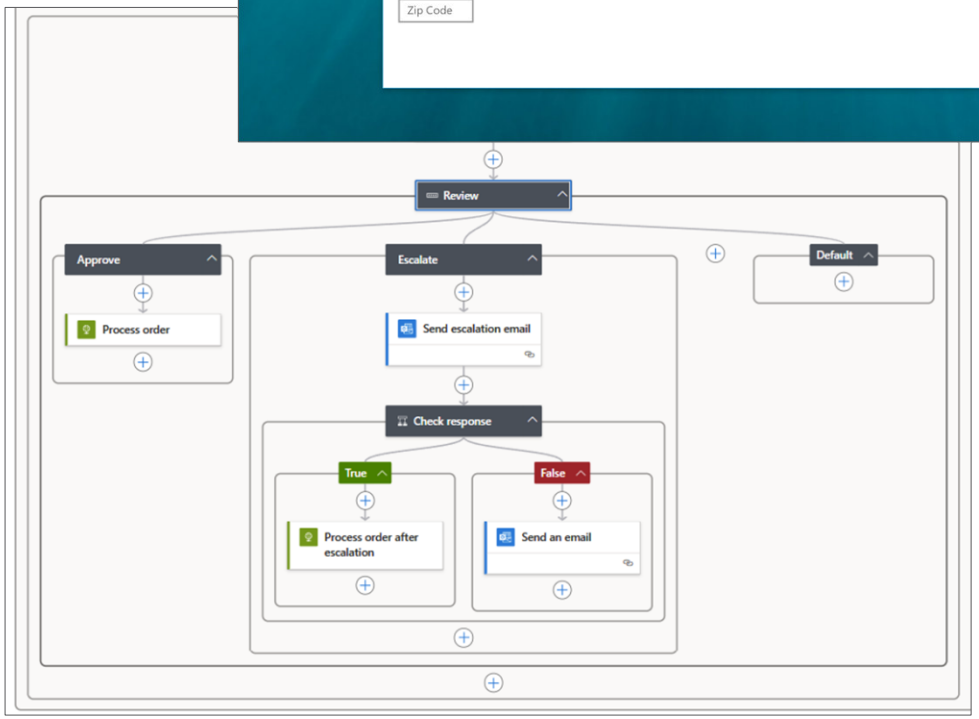
NKD

MOTIVATION

What if we could automate app deployment and configuration?
Or, how the NKD data science teams went from awkward to awesome

- Starting point
  - Development in Jupyter notebooks on dev machines
  - Most apps run on request from business users
  - Interactive apps using MS Forms and Logic apps
    - No version control
    - Not file based
    - Hard to parametrize
    - Hard to backup and setup after incident
    - Hard to modularize
    - Hard to debug
    - Outdated user experience





MOTIVATION

What if we could automate app deployment and configuration?
Or, how the NKD data science teams went from awkward to awesome

- Step by step
  - Common coding rules, using a linter and formatter
  - Work with repositories
  - Integrate common code into utility library
  - Build PoC web app with streamlit
  - Learn about pipelines
- Final stage
  - Common development environment as default
  - Template app, based on internal utility library
  - Central pipeline repository with versioning

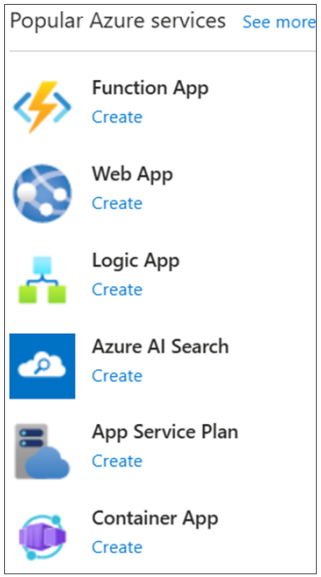
Streamlit

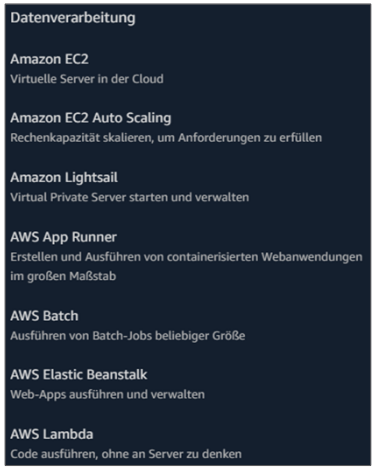
[1] https://streamlit.io/

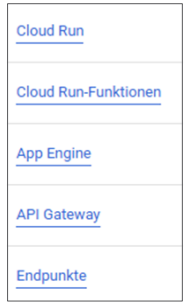
CHALLENGES

- Collaboration between developers
- Virtual environment and dependencies across different systems

- Many different possible solutions to build data apps on cloud services
- Large variety of involved systems and technologies
  - Code repository
  - CI/CD pipeline
  - Authentication
  - Authorization
  - Backend
  - Frontend
  - Web server
  - Database
  - File storage
  - Resource permissions
  - Package / artifact index
  - Secret management
  - Logging / debugging
  - Alerts
  - Monitoring
  - …
- Complex details make it hard to get it right
  - Connecting services
  - Many config options
  - Not total control over systems
  - Pipeline delay during development







- When not actively developing pipeline, knowledge is lost again
- Updates to the pipeline during project development leads to fragmented work, which must be re-done multiple times

TOOLS

- Dependency Manager – poetry
- Code Formatter and Linter – ruff
- Cloud Resource Management – Azure CLI
- Project Configuration – pyproject.toml `pyproject.toml`

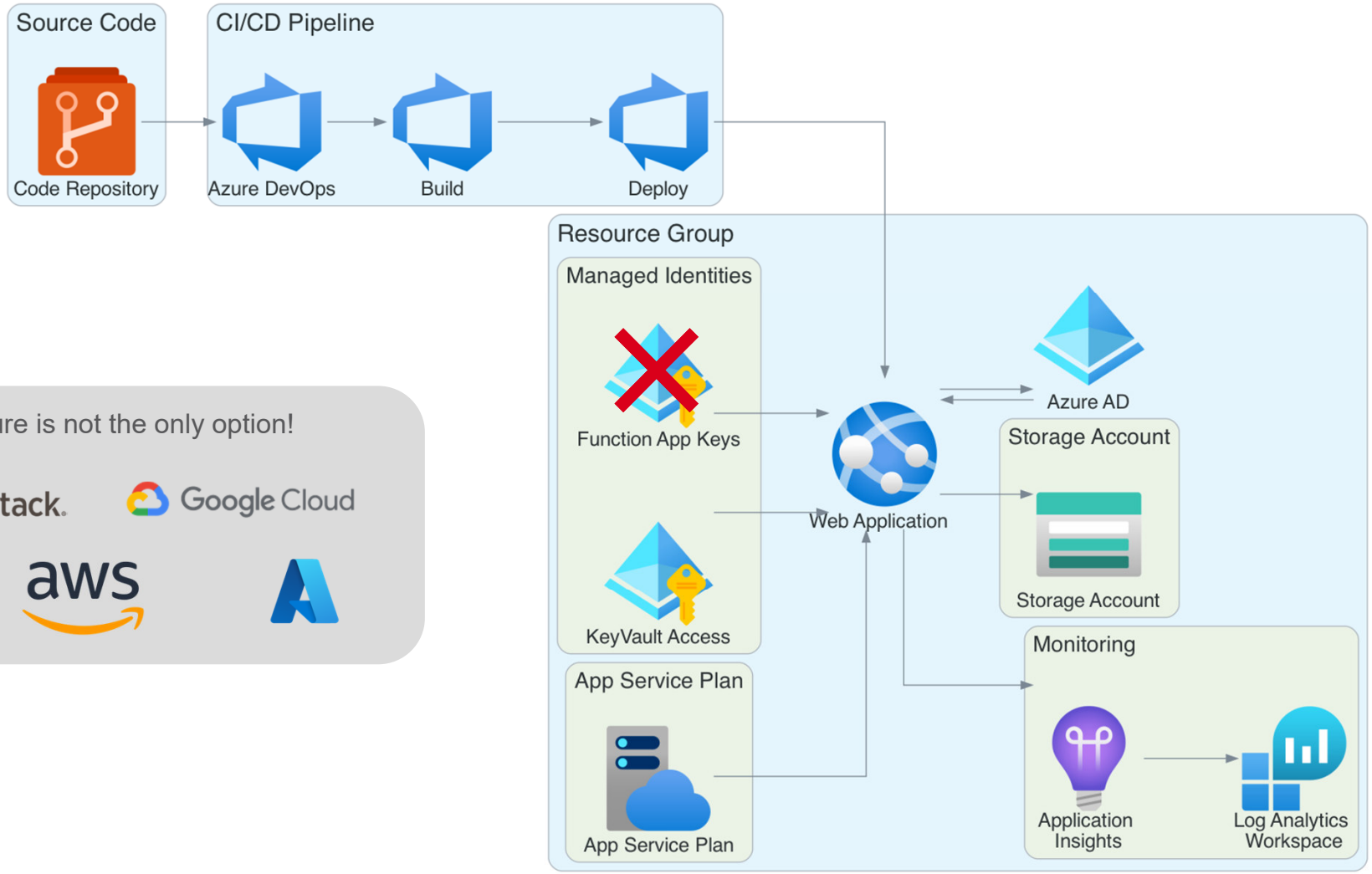
More details in this talk

https://github.com/JSchoeck/talk_python_ruff-poetry
2024 @ Python User Group Nürnberg

```toml
[project]
# TODO: replace with actual project name, version, description, and author(s)
name = "TEC_002_Web_App_Template"  # Name must have this structure: [type in 3 letters: TEC/PAS/PRO]
version = "2.0.0"
description = "This is a template for Python projects that are hosted as an Azure App Service (i.e.
authors = [
  {name = "Johannes Schöck", email = "Johannes.Schoeck@nkd.de"},
  {name = "Luis Cuervo", email = "Luis.Cuervo@nkd.de"},
  {name = "Sai Nallamothu", email = "SaiRam.Nallamothu@nkd.de"},
]
readme = "README.md"
requires-python = ">=3.11"
dependencies = []

[tool.poetry]
package-mode = false
```

[1] https://python-poetry.org/; [2] https://docs.astral.sh/ruff/; [3] https://learn.microsoft.com/de-de/cli/azure/install-azure-cli;
[4] https://packaging.python.org/en/latest/guides/writing-pyproject-toml/

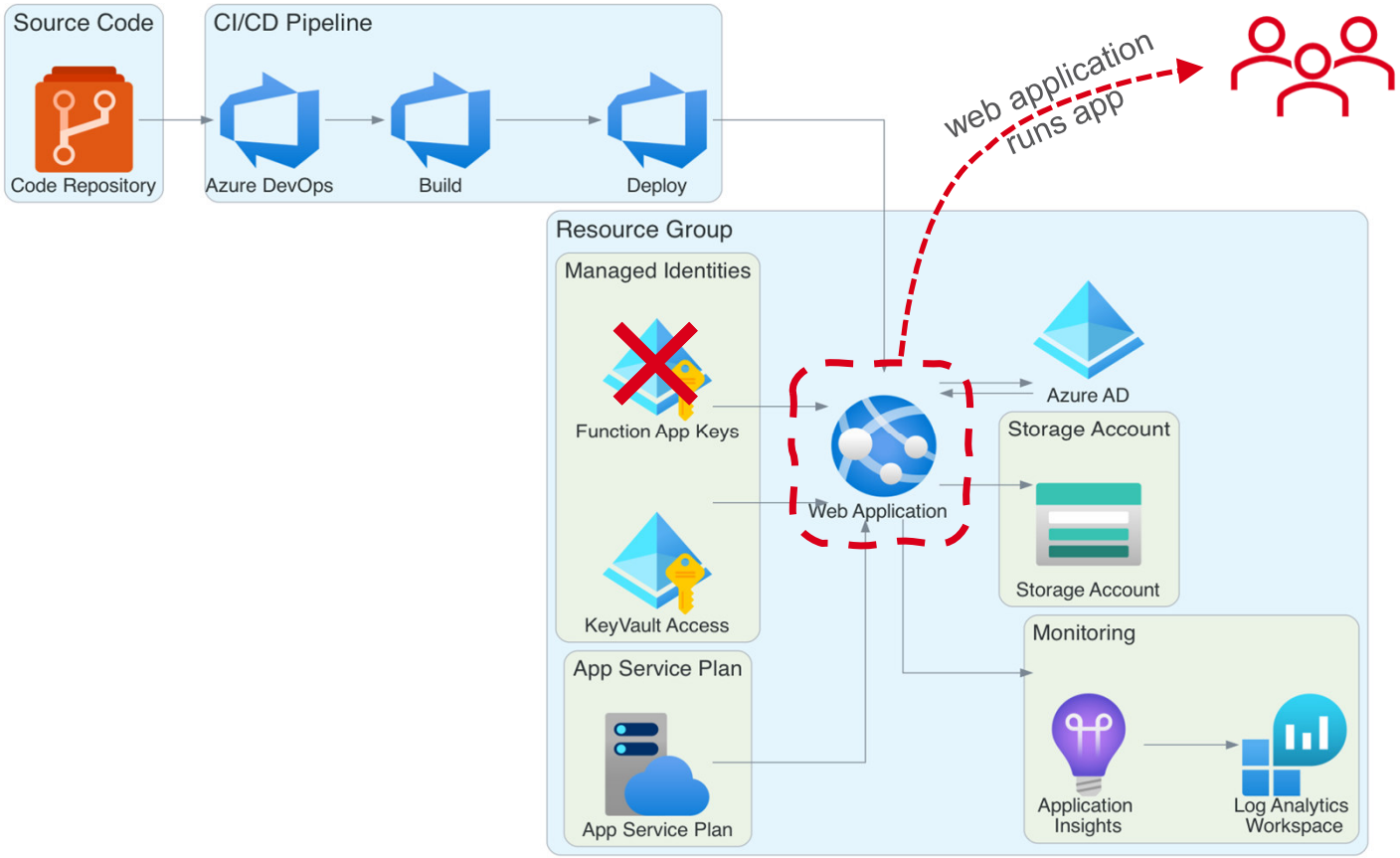
AZURE INFRASTRUCTURE



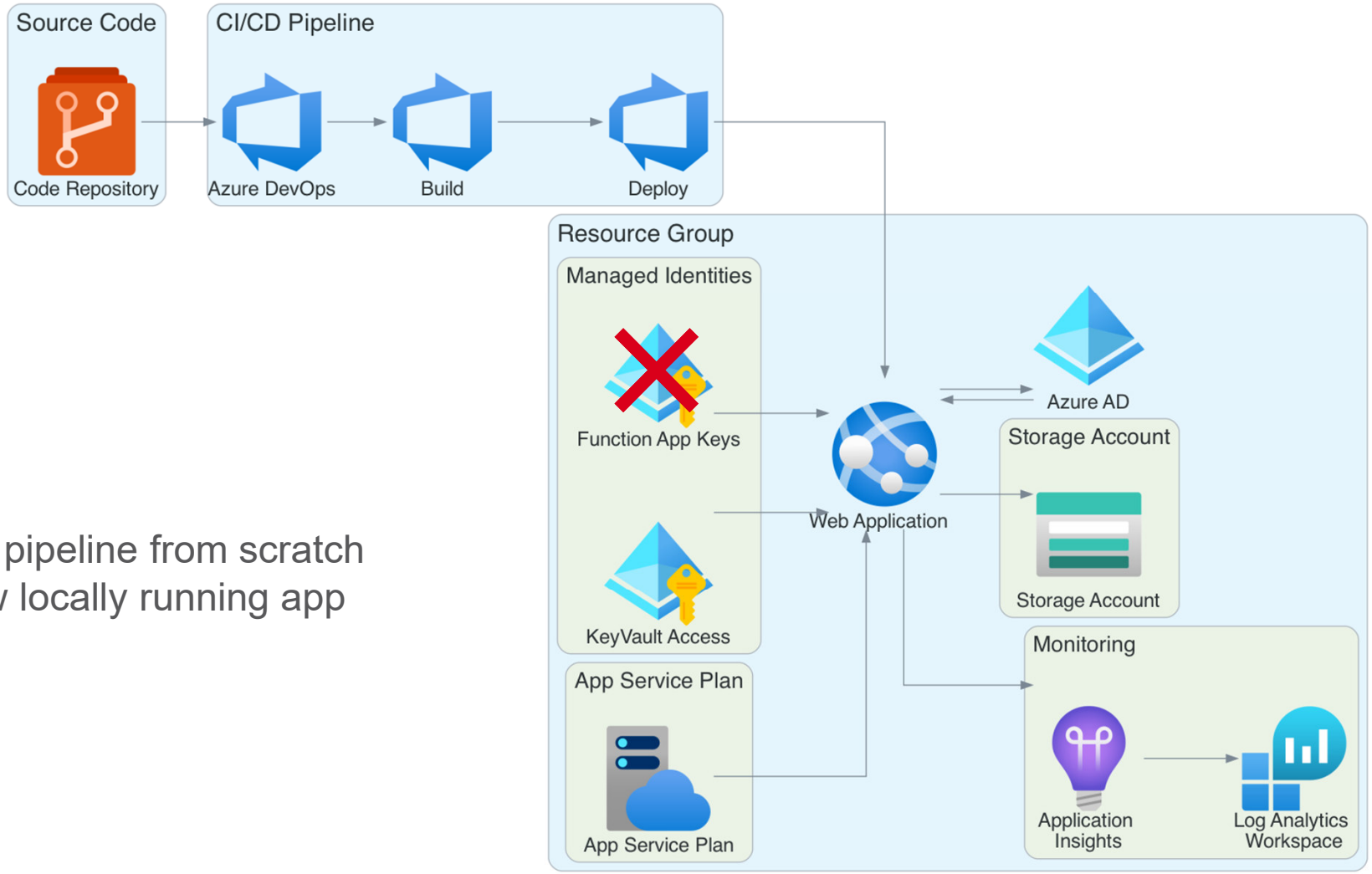
Azure is not the only option!

openstack. Google Cloud aws

AZURE INFRASTRUCTURE



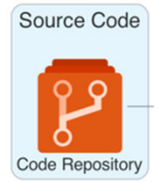
SHOWCASE



Live:

1. Start pipeline from scratch
2. Show locally running app

THE TEMPLATE – APP DEVELOPMENT & CONFIGURATION



- Poetry configuration in poetry.toml

```toml
[virtualenvs]
in-project = true

[virtualenvs.options]
always-copy = true
```

- Streamlit configuration in .streamlit\config.toml

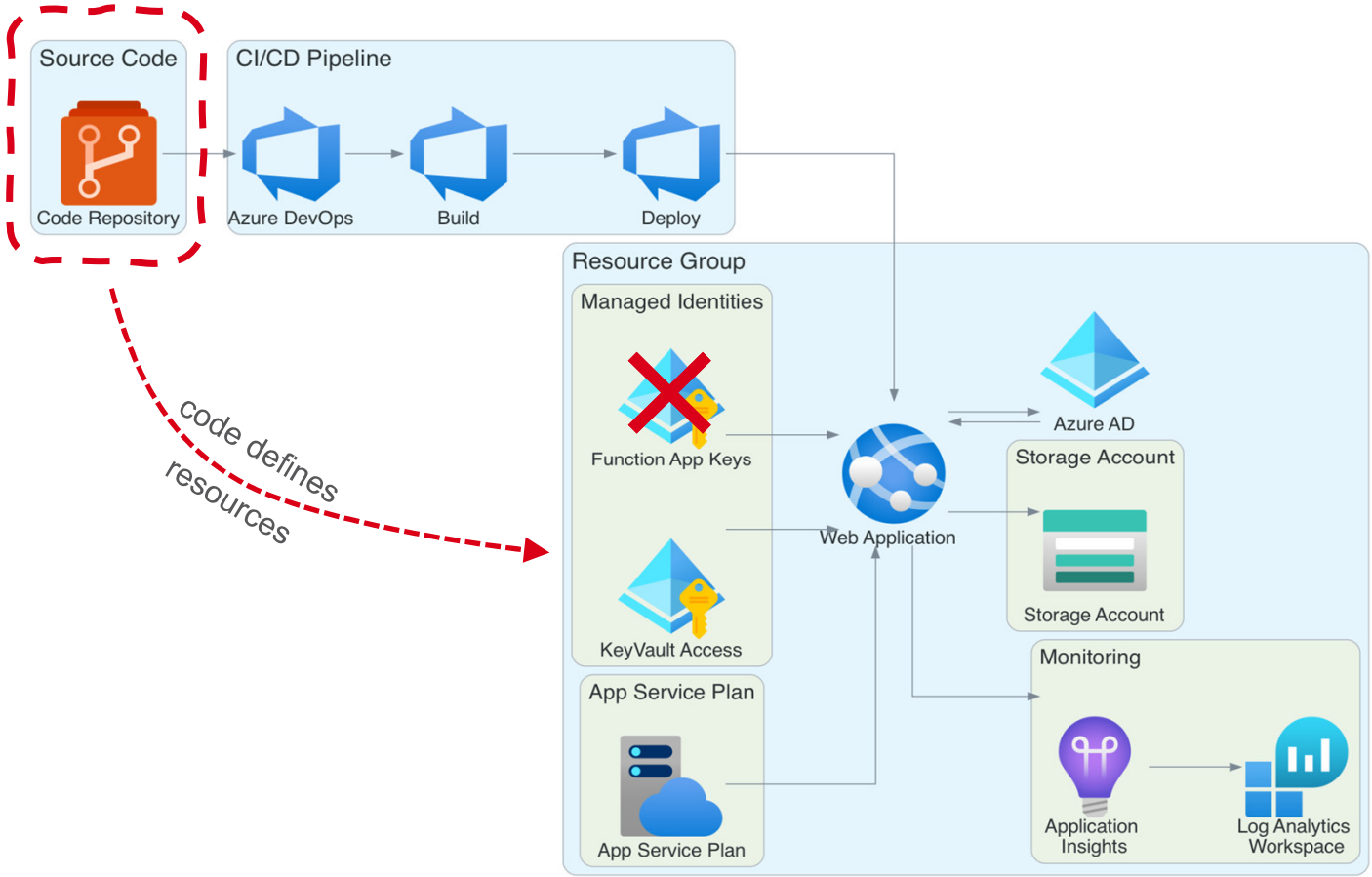
```toml
[theme]
base="light"
primaryColor="#F63366"
backgroundColor="#FFFFFF"
secondaryBackgroundColor="#FADEDE"  # production: F0F2F6; development: FAD
textColor="#262730"
font="sans-serif"

[client]
toolbarMode="minimal"  # https://docs.streamlit.io/develop/concepts/archit
showSidebarNavigation = false  # Controls whether the default sidebar page
# Alternatively build custom navigation: https://docs.streamlit.io/develop

[browser]
gatherUsageStats = false  # always set to false, otherwise user data will
```

- Running & debugging streamlit

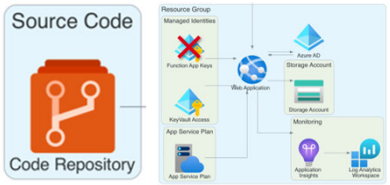

```jsonc
{
  "version": "0.2.0",
  "configurations": [
      {
          "name": "Streamlit app.py",
          "type": "debugpy",
          "request": "launch",
          "module": "streamlit",
          "env": {
              "STREAMLIT_APP": "\\src\\app.py",
              "PYTHONPATH": "${workspaceRoot}",
              "PYDEVD_WARN_SLOW_RESOLVE_TIMEOUT": "30"
          },
          "args": [
              "run",
              "${workspaceRoot}\\${STREAMLIT_APP}"
          ],
          "redirectOutput": true,
          "preLaunchTask": "prepare environment",
      },
  ]
}
```

```jsonc
{
    "label": "poetry install",
    "type": "shell",
    // Install dependencies from lock file into environment, without installing
    "command": "poetry install --no-root",
    "dependsOn": "poetry lock refresh"
}
```

AZURE INFRASTRUCTURE



THE TEMPLATE – DEPLOY.BICEP



Defines what resources get created

1. Fetch KeyVault ID, to avoid hardcoding it

```
resource keyVault 'Microsoft.KeyVault/vaults@2023-07-01' existing = {
  scope: resourceGroup(subscriptionIdProd, utilitiesResourceGroupName )
  name: 'kv-datascience-general'

}
```

2. Create new resource group, with standardized name

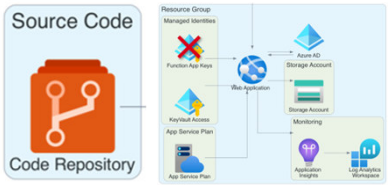
```
resource newRG 'Microsoft.Resources/resourceGroups@2022-09-01' = {
  name: 'rg-${projectId}-${projectName}'
  location: location
  tags: {
    owner: 'data science'
  }
}
```

3. Pass parameters to resource creation bicep file

```
module resources 'resources.bicep' = {
  name: 'resources'
  scope: newRG
  params: {
    projectId: projectId
    projectName: projectName
    location: location
    utilitiesResourceGroupName: utilitiesResourceGroupName
    pathToApp: pathToApp
    pythonVersion: pythonVersion
    runFromPackage: runFromPackage
    blobStorage: blobStorage
    skuProd: skuProd
    skuDevtest: skuDevtest
    existingASPname: existingASPname
    miGeneralKeyvaultAccess: dsManagedIdentityKeyvaultAccessClientId
    subscriptionIdProd: subscriptionIdProd
  }
}
```


NKD

THE TEMPLATE – RESOURCES.BICEP



1. Define default parameters
2. Set log analytics workspace to existing shared workspace
3. Set app service plan (ASP) to existing shared compute resource
4. “Optional”: create storage account, blob service, and blob container and set accessibility
5. Create application insights, targeting log analytics workspace
6. Create ASP, if not using the shared one
7. Create web application
   - Assign managed IDs to access KeyVaults
   - Set startup command for streamlit
   - Define environment variables to connect services and configure app
8. Create basic blocking identity provider

Compute & costs

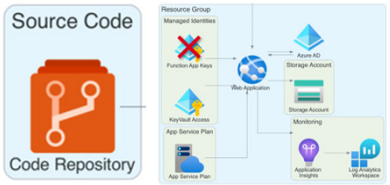
- ASPs have monthly fixed cost
- Web apps are not in active use most of the time
- More CPU and RAM → faster app flow
- Sharing compute resources allows high performance with low cost

Naming

- Resources have fixed naming schemes
- Split into production and development environment is represented in most resource names by adding “-dev”

```
1  resource webApplication 'Microsoft.Web/sites@2024-04-01' = {
2    name: environment == 'production' ? 'wa-${projectFullName}' : 'wa-${projectFullName}-dev'
```

THE TEMPLATE – RESOURCES.BICEP



1. Define default parameters
2. Set log analytics workspace to existing shared workspace
3. Set app service plan (ASP) to existing shared compute resource
4. “Optional”: create storage account, blob service, and blob container a
5. Create application insights, targeting log analytics workspace
6. Create ASP, if not using the shared one
7. Create web application
   - Assign managed IDs to access KeyVaults
   - Set startup command for streamlit
   - Define environment variables to connect services and configure ap
8. Create basic blocking identity provider

Resource interdependence

- Some resources depend on the existence of others
- Creation order can be enforced using *dependsOn*

```
dependsOn: empty(existingASPname) ? [appServicePlan] : [existingAppServicePlan]
```

- Settings to allow resource access to other components are stored as application variables in the web app component

```
{
  name: 'AZURE_STORAGE_ACCOUNT_NAME'
  value: blobStorage ? storageAccount.name : ''
}
```

Streamlit configuration

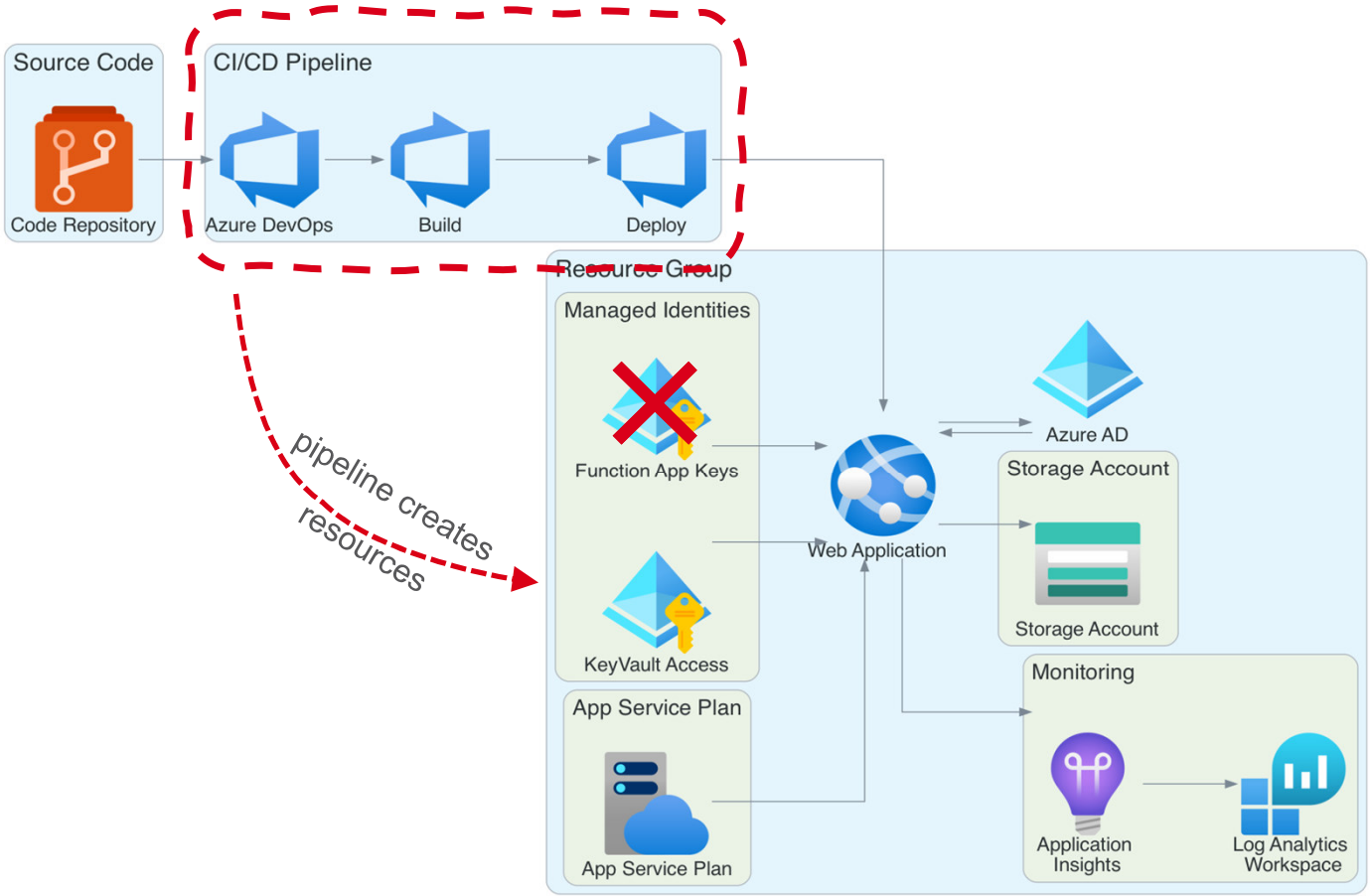
Streamlit

- Start-up command needs to be customized, as only Django & Flask support is integrated
- XSRF Protection needs to be disabled, otherwise Azure components from different servers cannot interact properly (authentication, storage)

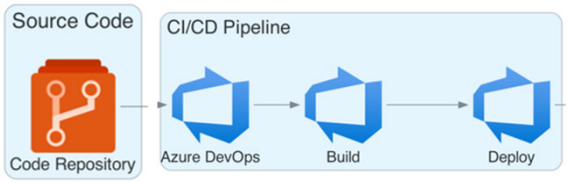
```
appCommandLine: 'python${pythonVersion} -m streamlit run /home/site/wwwroot/${pathToApp} --server.port 8000 --server.address 0.0.0.0 --server.enableXsrfProtection=false'
```

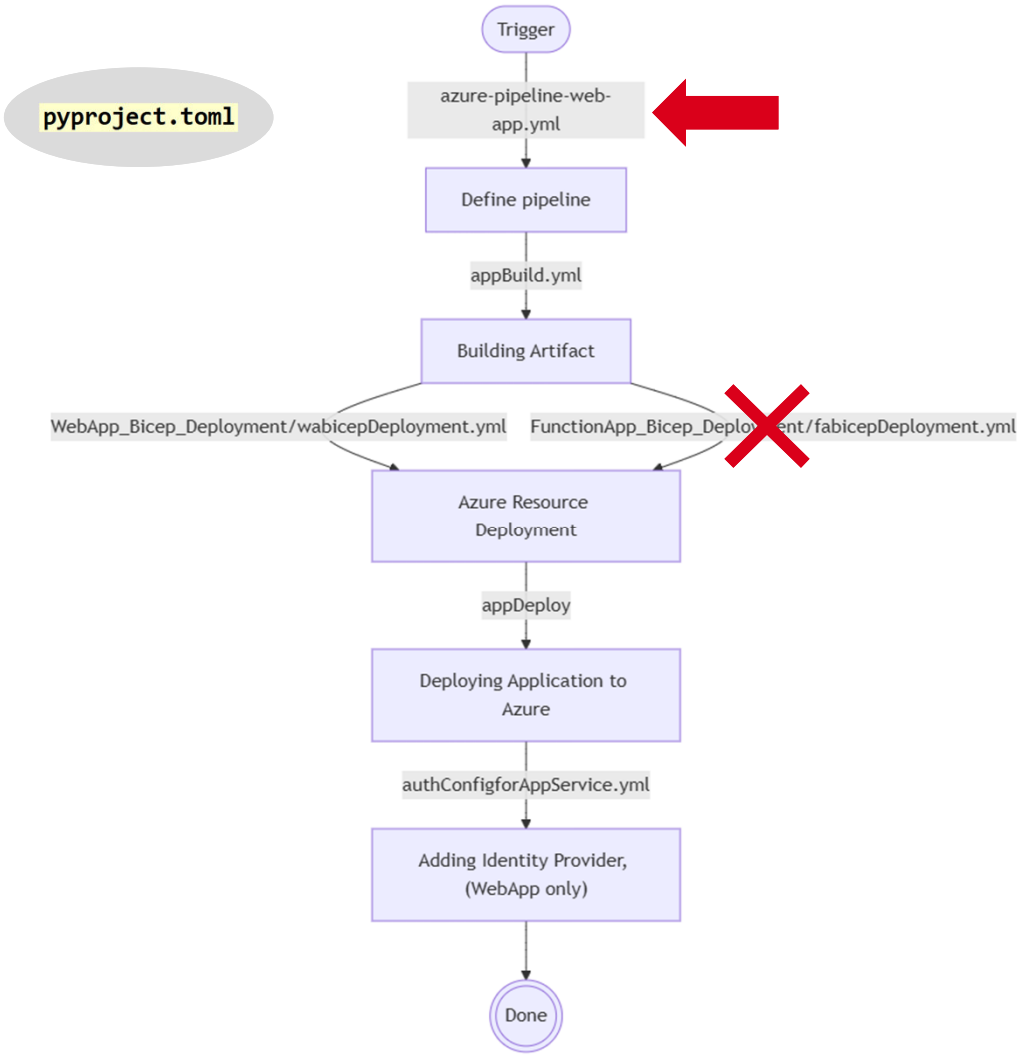

NKD

AZURE INFRASTRUCTURE

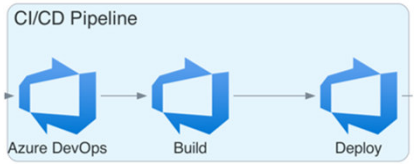


THE PIPELINE – STRUCTURE OF THE STAGES





THE PIPELINE – AZURE-PIPELINE-WEB-APP.YML



Declaration of pipeline stages

1. **Resources**:
   Define repository for the centralized pipeline template files

2. **Trigger:**
   Define default triggers for main and development branch
   Optional: include / exclude paths, files, or file types

3. **Variables**:
   Import Azure DevOps pipeline variable groups
   Set constants and initiate variables

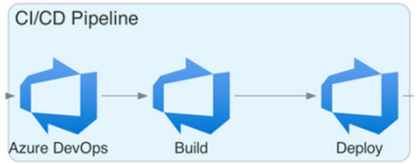
4. **Stages**:
   Defines stages of the pipeline, which template pipeline file to use for it, and which parameters to pass to each stage

```yaml
variables:
  - name: isMain
    value: $[eq(variables['Build.SourceBranchName'], 'main')]
```

→ Matching branch and target server

```yaml
stages:
- template: appBuild.yml@Shared_Templates
```

THE PIPELINE – PARAMETERS



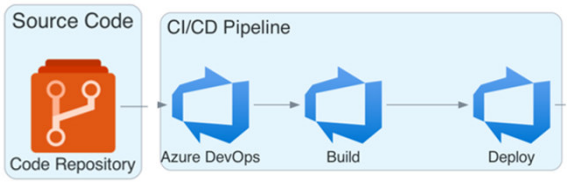
Minimizing manual configuration with file parsing and bicep

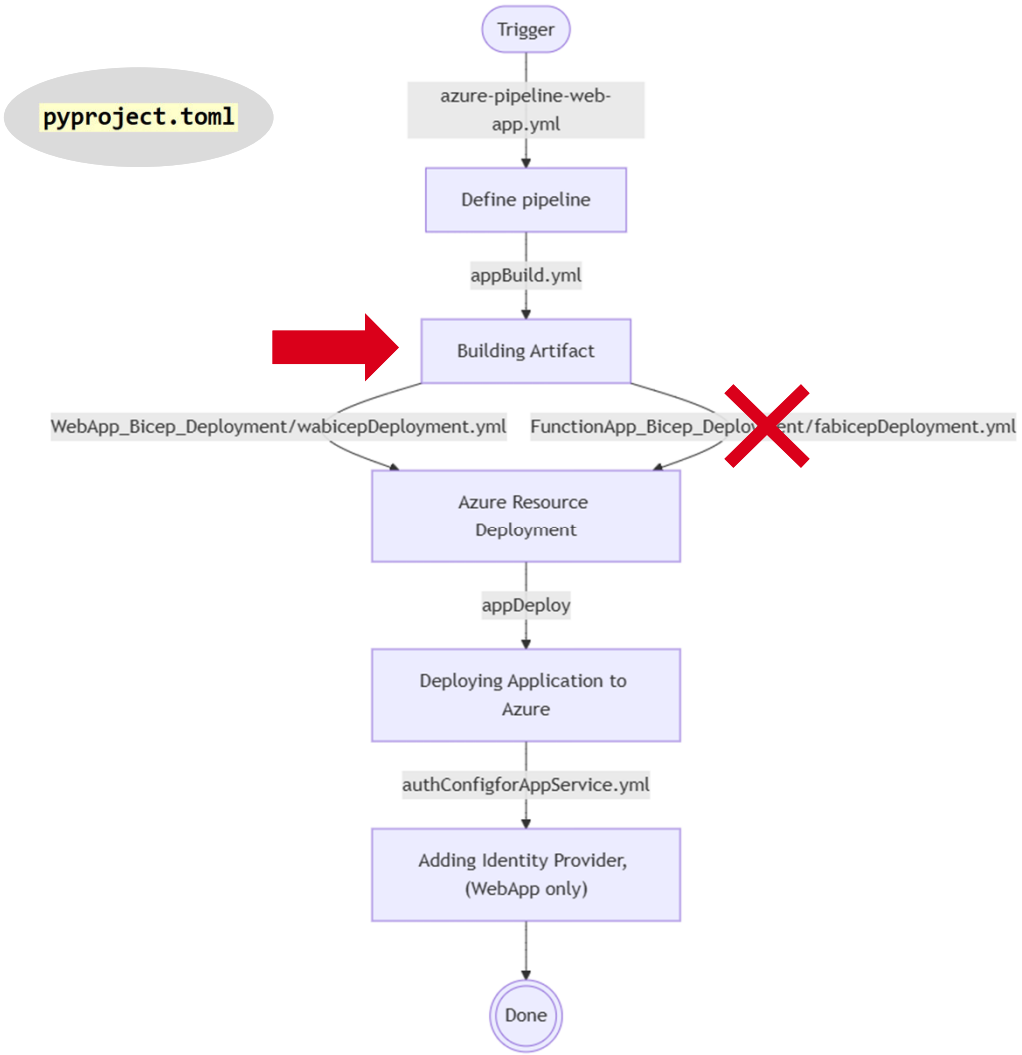
- Azure resources already defined in the bicep template
- But the pipeline also needs some information
- Solution:
  - Parse parameters from our one and only source of adaptable configuration: `pyproject.toml`
  - Modifications and error-handling trivial to implement
  - Set parameters as environment variables in the pipeline

ado_pipeline_repo\WebApp_Bicep_Deployment\deploy\nkd_pipeline_utils.py

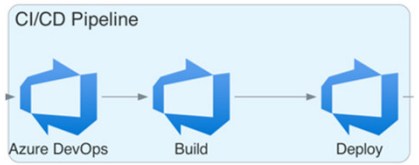
```python
def get_var_from_pyproject(var_name: str, path: Path | None = None) -> None:
    if path is None:
        path = Path.cwd().joinpath("pyproject.toml")
    config = toml.load(path)
    if var_name == "pythonVersion":
        var_value = clean_version_string(config["project"]["requires-python"])
    elif var_name == "nkd_utils_lib_version":
        var_value = clean_version_string(config["tool"]["poetry"]["dependencies"]["nkd-utils-lib"]["version"])
    else:
        print(f"Error: Variable {var_name} not found in pyproject.toml file.")
        var_value = None
    set_pipeline_variable(var_name, var_value)
```

THE PIPELINE – STRUCTURE OF THE STAGES





THE PIPELINE – APPBUILD.YML

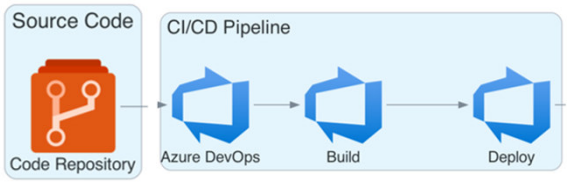


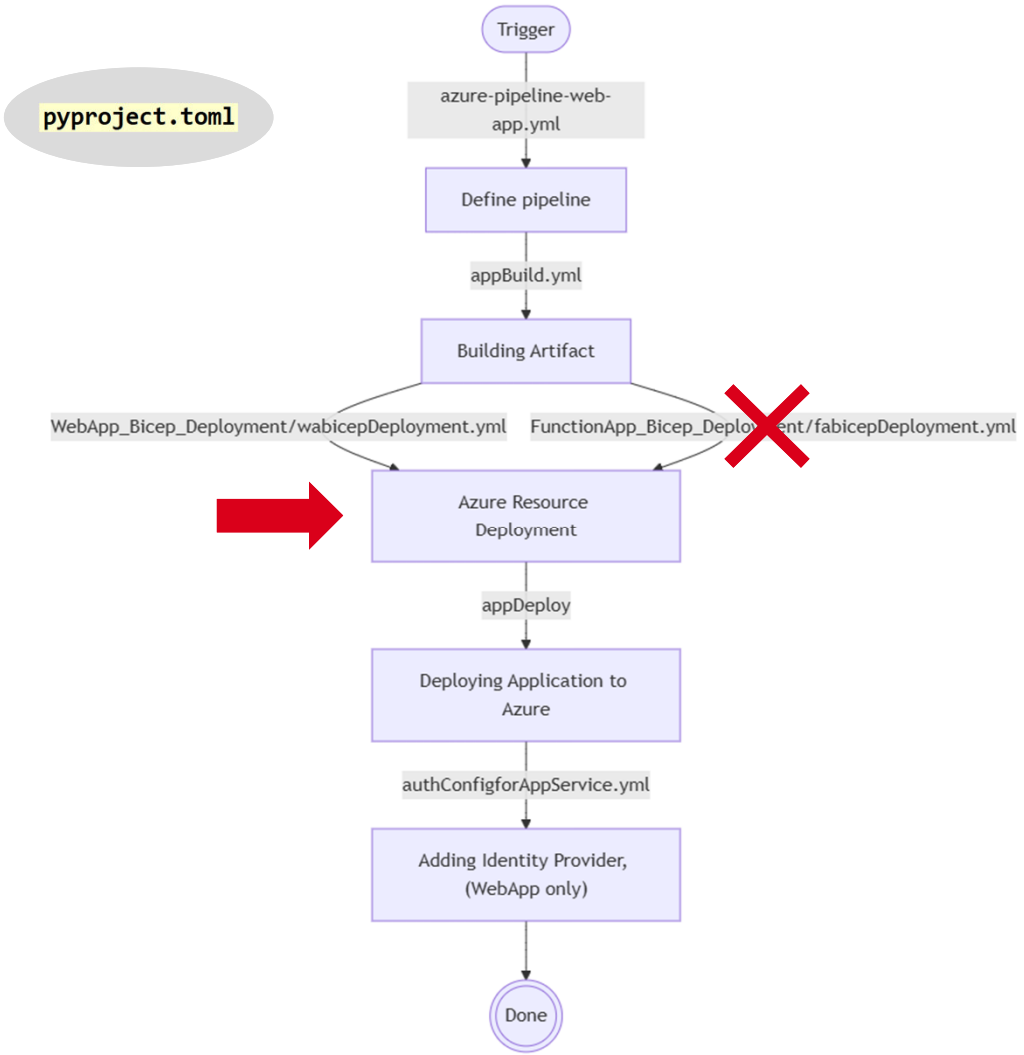
Steps of build stage

1. Find pyproject.toml and set path as environment variable
2. Extract project variables to set them globally
3. Clean up pipeline template and structure app folder
4. Authenticate to Azure DevOps artifact storage
5. Install poetry
6. Update lock file and export dependencies
7. Install requirements
8. Remove unneeded files
9. Create zip archive from virtual environment and web app files
10. Upload zip file from build to pipeline server

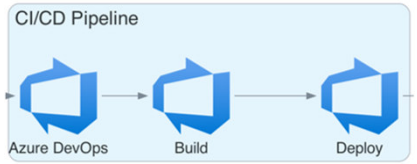
- Exporting to requirements.txt and pip installing from there to handle missing native support for poetry
- Installation of requirements into target environment‘s *site-packages* folder
- Deleting files that are not required to run the web app decreases deployed file size and avoids accidental publication

THE PIPELINE – STRUCTURE OF THE STAGES





THE PIPELINE – WABICEPDEPLOYMENT.YML

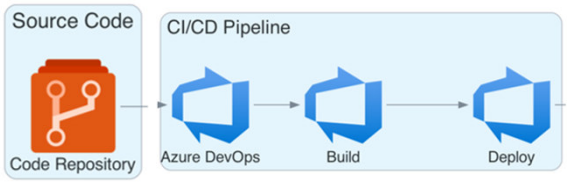


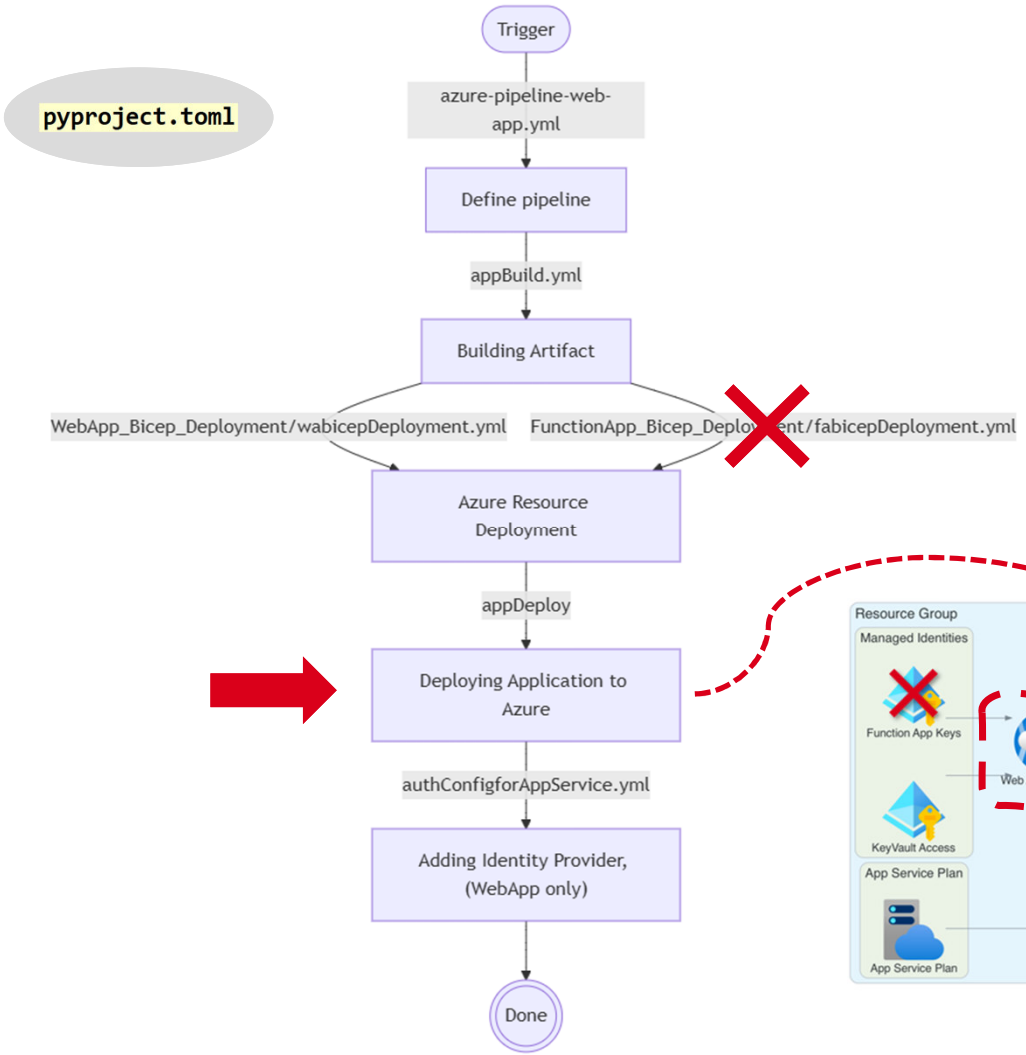
Steps of Azure resource creation stage

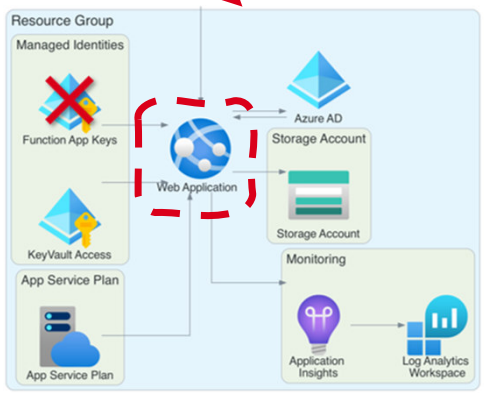
1. Define variables for pipeline (most from params)
2. Read subscription ID and put it in variable
3. Download bicep template files
4. Set override parameters from app config
5. Deploy Azure Resources based on bicep template

- Define all parameters to receive from calling pipeline
- Starting the stage depends on success of previous build stage
- Resource definition happens in .bicep file
- Default settings for resources are defined in .bicepparam file
- Azure CLI can be used in pipeline bash scripts

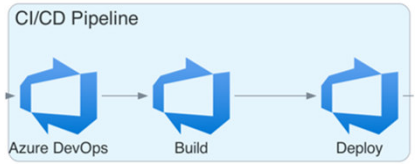
THE PIPELINE – STRUCTURE OF THE STAGES







THE PIPELINE – APPDEPLOY.YML



Steps of deployment stage

1. Define variables for pipeline
2. Deploy zip artifact with AzureWebApp task

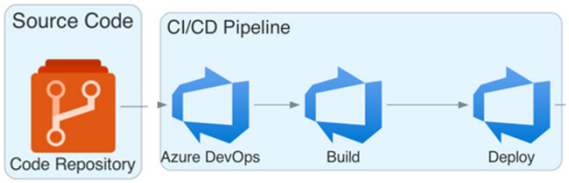
```yaml
steps:
- task: AzureWebApp@1
  displayName: 'Deploy Azure Web App'
  inputs:
    azureSubscription: ${{ variables.armconnection }}
    appName: ${{ parameters.appName }}
    appType: 'webAppLinux' # $(appType) it should be always webAppLinux
    package: $(Pipeline.Workspace)/drop/$(Build.BuildId).zip
```

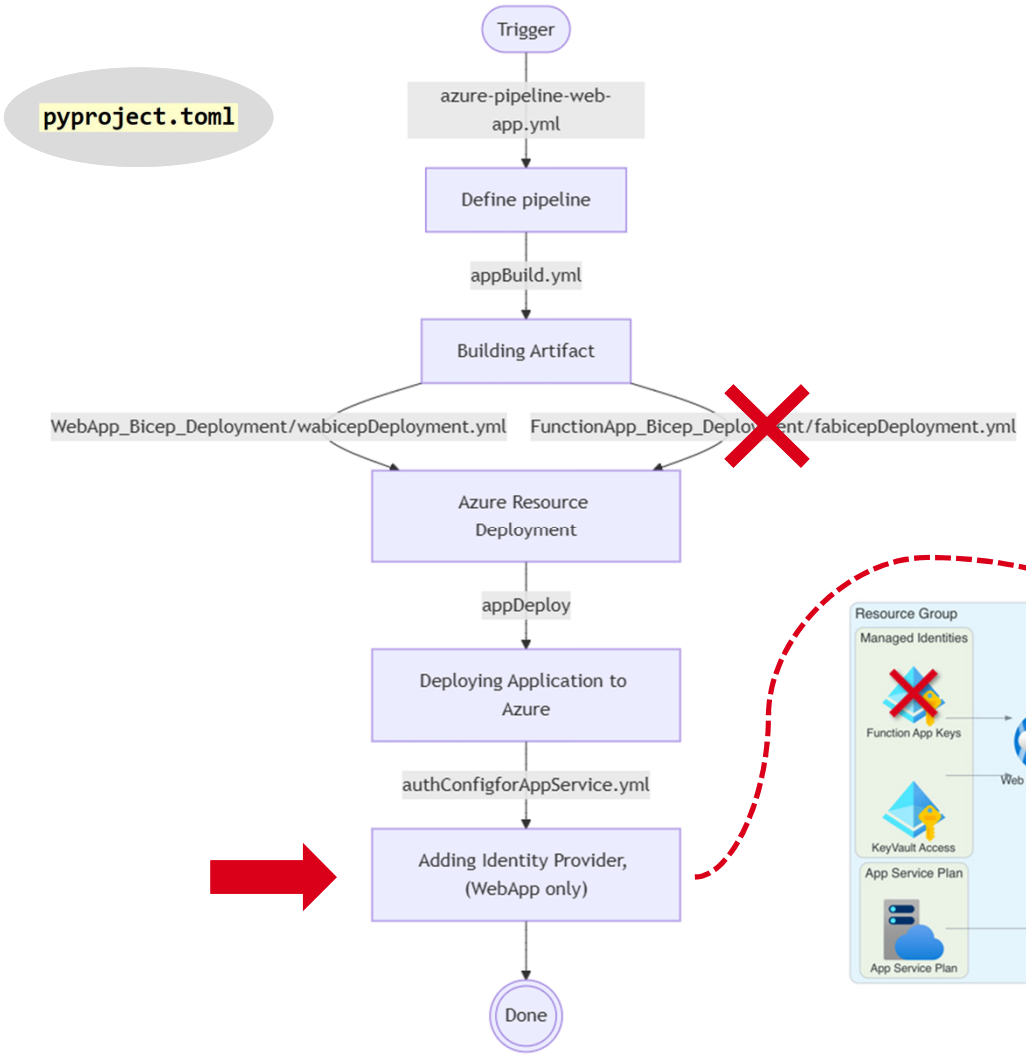
- Define all parameters to receive from calling pipeline
- Starting the stage depends on success of all previous stages

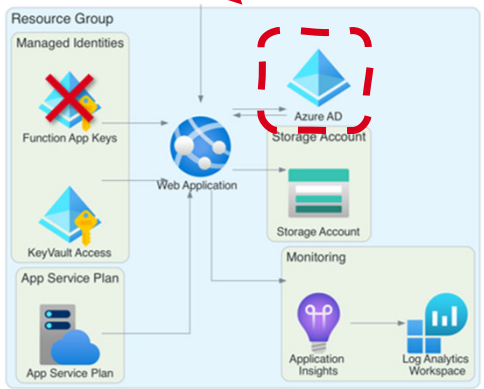
```yaml
parameters:
  - name: armconnection
    type: string
  - name: appName
    type: string
  - name: appType
    type: string
  - name: projectId
    type: string
  - name: projectName
    type: string

stages:
- stage: DeployApp
  displayName: 'Deploy to azure environment'
  dependsOn:
    - Build
    - DeployBicep
  variables:
    armconnection: ${{ parameters.armconnection }}
    appName: ${{ parameters.appName }}
    appType: ${{ parameters.appType }}
    isMain: ${{ variables.isMain }}
    projectId: ${{ parameters.projectId }}
    projectName: ${{ parameters.projectName }}
```


NKD

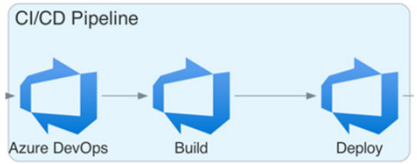
THE PIPELINE – STRUCTURE OF THE STAGES







THE PIPELINE – AUTHCONFIGFORAPPSERVICE.YML

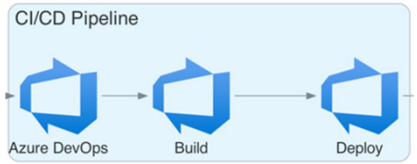


Steps of authorization configuration stage

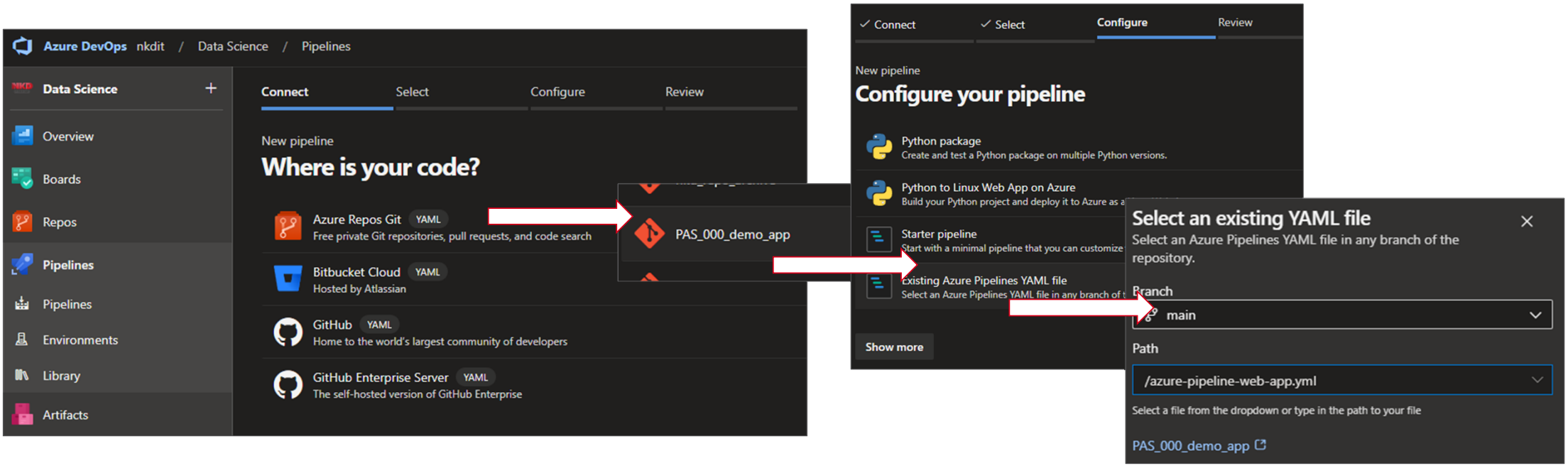
1. Define variables for pipeline (from params)
2. Execute Azure CLI script on pipeline server
    1. Check if **app registration** exists in subscription
    2. If it does not yet exist, create app registration
    3. Add client secret from Azure DevOps
    4. Add authentication extension to Azure CLI
    5. **Limit access** to users from same tenant
    6. Enable and configure web app authentication
    7. Assign app registration to web app
    8. Configure app service to only accept connections from matching app registration
    9. If it does not yet exist, create **service principal** for app registration
    10. Set authorization by adding users / groups to service principal

- Define all parameters to receive from calling pipeline
- Starting the stage depends on success of all previous stages

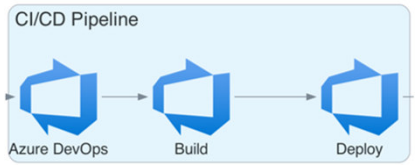
THE PIPELINE – CREATION IN ADO



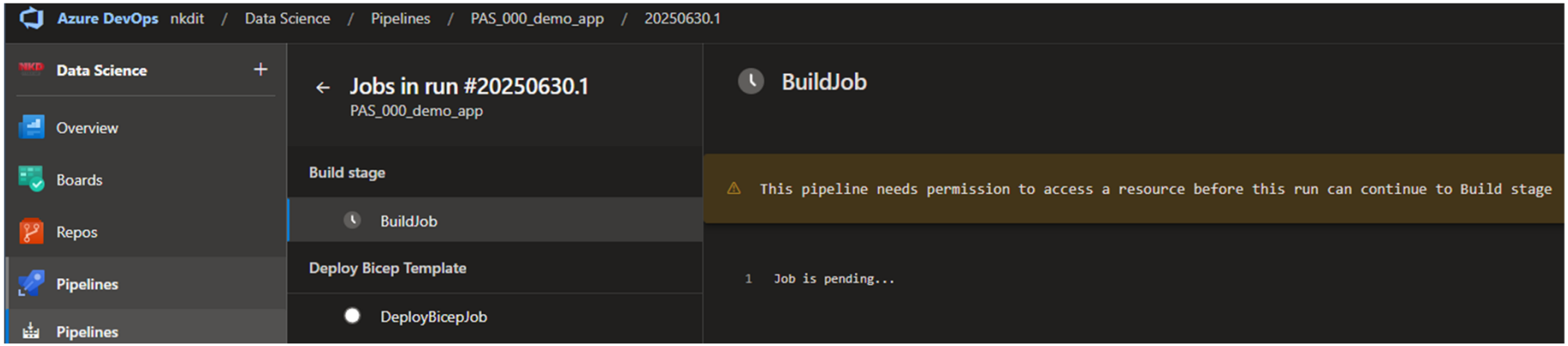
- Manual pipeline creation – required once
  - Pipelines → New pipeline → Azure Repos Git
  - Select repository with app code
  - Existing Azure Pipelines YAML
  - Select azure-pipeline-web-app.yml

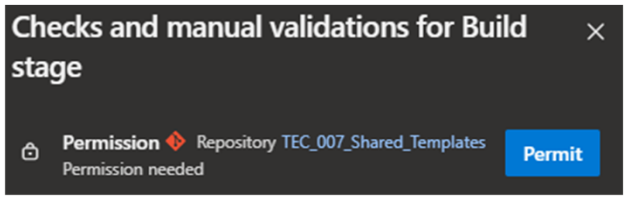


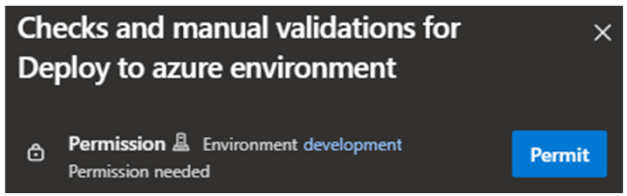
THE PIPELINE – CREATION IN ADO

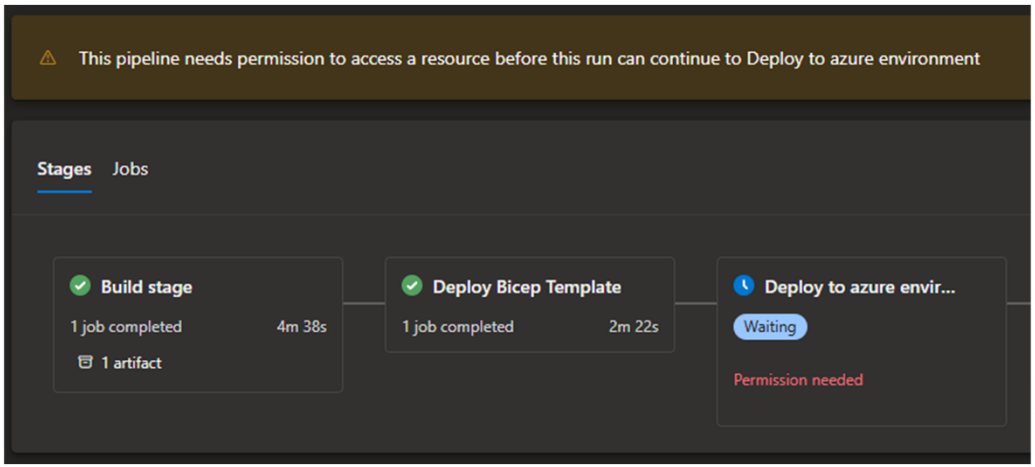


- After starting first pipeline run, pipeline stops… 😖
- Permission required to access centralized pipeline repository and Azure “environment”





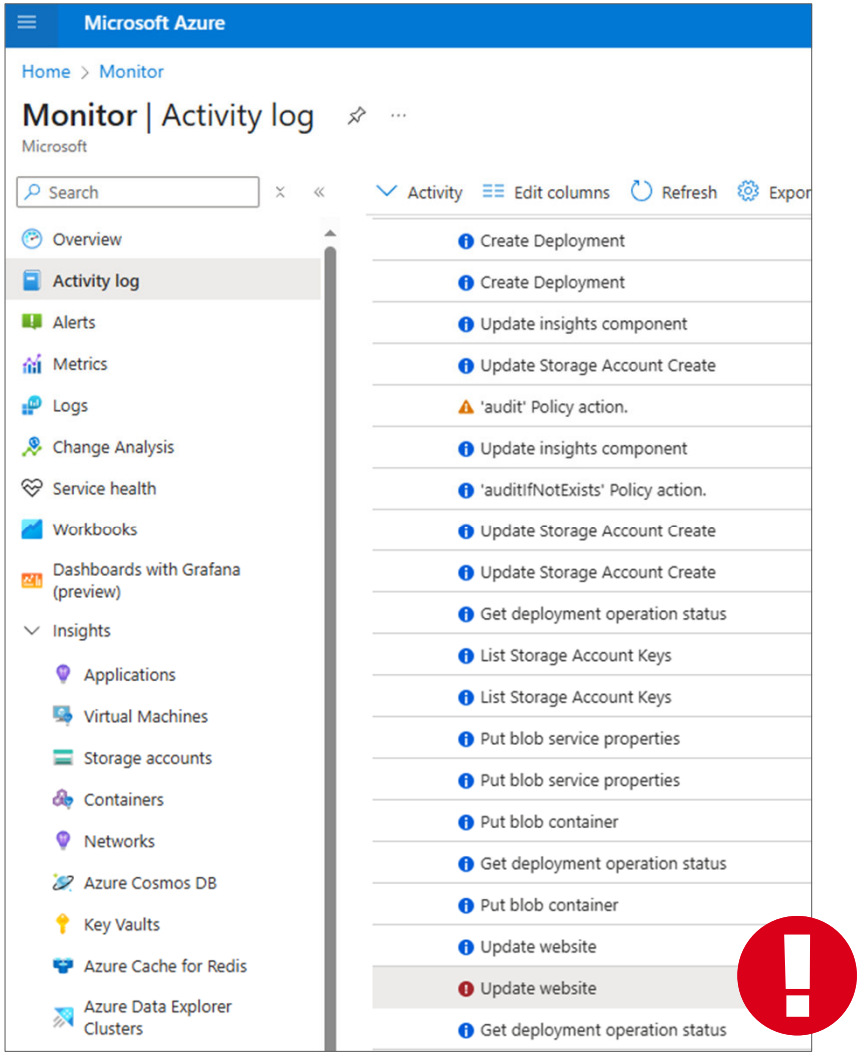




THE PIPELINE – DEBUGGING THE DEPLOYMENT

- Debugging deployment errors:

  Azure Monitor – Activity Log

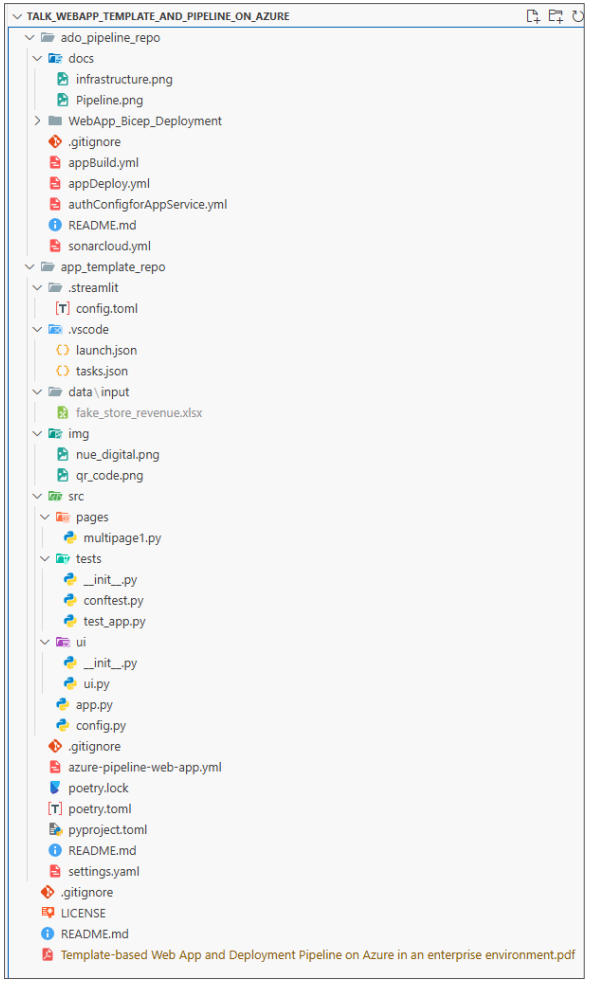


SHOWCASE

- Going back to the live running pipeline, we hopefully see this:

Deploy Azure Web App

```
Starting: Deploy Azure Web App
==============================================================================
Task         : Azure Web App
Description  : Deploy an Azure Web App for Linux or Windows
Version      : 1.257.0
Author       : Microsoft Corporation
Help         : https://aka.ms/azurewebapptroubleshooting
==============================================================================
Got service connection details for Azure App Service:'wa-PAS-000-demo-app'
Package deployment using ZIP Deploy initiated.
Deploy logs can be viewed at https://wa-pas-000-demo-app.scm.azurewebsites.net/api/deployments/4ec51e29-7f6c-49b8-a394-d80cdd1f15dc/log
Successfully deployed web package to App Service.
Successfully updated App Service configuration details
Successfully added release annotation to the Application Insight : wa-PAS-000-demo-app-appinsights
Successfully updated deployment History at https://wa-pas-000-demo-app.scm.azurewebsites.net/api/deployments/231381751290632708
App Service Application URL: https://wa-pas-000-demo-app.azurewebsites.net
Finishing: Deploy Azure Web App
```



https://wa-pas-000-demo-app.azurewebsites.net/

YOU WANT MORE?

- Setup script
  - Ask user questions about new project
  - Copy template app folder
  - Create local virtual environment (in .venv)
  - Create repository
- Automate pipeline creation
  - Authorization to use pipeline template repo in pipeline
  - Adding pipeline to new project repository

````
```bash
curl -u :<PAT> \
-X POST \
-H "Content-Type: application/json" \
-d '{
    "folder": "\\",
    "name": "MyPipeline",
    "configuration": {
        "type": "yaml",
        "path": "azure-pipelines.yml"
    }
}' \
https://dev.azure.com/{organization}/{project}/_apis/pipelines?api-version=6.0-preview.1
```
````

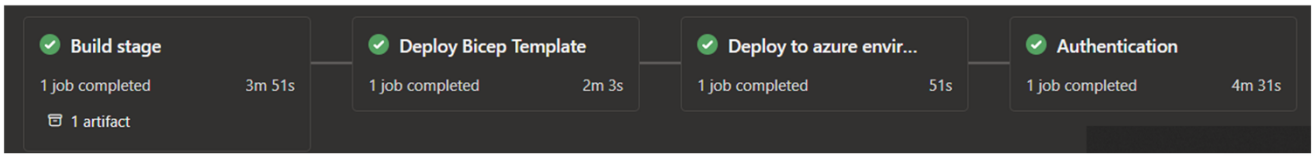
- Other ideas:
  - Enforce formatting and linting via pre-commit
  - Test scripts
  - Replace anaconda & poetry with uv
- Already implemented:
  - Deploy Azure Function Apps with same code base
  - Code quality and security checks with SonarCloud
  - Common development environment in VS Code
    - Linting
    - Formatting
    - Virtual environment with important dependencies
    - Install tools (Azure CLI, poetry, VS code extensions)
    - Authentication to Azure DevOps artifact feed

NKD

LESSONS LEARNED

- Standardize early

- Use a small selection of tools with as few configuration files as possible
- Create and develop app and pipeline templates in centralized and shared repositories, including release versioning
- Archive (zip) deployment is a must, otherwise deployment time explodes (5 vs. 45 minutes)
  - But: its read-only file system requires using external storage (e.g., blob storage account on Azure)
- Pipeline environment variables can only be used in the step *after* they were set and are lost between blocks, unless set correctly
- There is more than one virtual environment on the web app server, which can lead to bugs from Azure components, accidentally overriding a different package version than what your app depends on
- In the pipeline use terminal commands for easy tasks and python scripts to handle more complex tasks
  python -c "from deploy import nkd_pipeline_utils; nkd_pipeline_utils.get_all_var_from_pyproject(path='$(pyprojectPath)')"
- The reward for learning these lessons:



QUESTIONS?

COMMENTS?

SUGGESTIONS?

Code and slides @GitHub

https://github.com/JSchoeck/talk_webapp_template_and_pipeline_on_azure

Find me on LinkedIn

THE TEMPLATE – APP DEVELOPMENT & CONFIGURATION

Tools that help with consistency and automation

Dependency Manager – poetry 

- Much faster dependency resolution than pip
- Lock file ensures 100% reproducibility
- Can authenticate with and use custom package indices like Azure DevOps artifacts
- Not fully compatible in Azure DevOps pipeline, but can export to requirements.txt → pip install

Code Formatter and Linter – ruff

- Ruff combines multiple other tools for code consistency (isort, black, Flake8, bandit, pydocstyle)
- Very fast (written in Rust, hardly noticeable for normal files)
- Common style avoids messy commits with many changes back and forth (e.g., single vs. double quotes)

Cloud Resource Management – Azure CLI 

- Cross-platform command-line tool
- Can manage and configure Azure resources
- Ideal for scripting automations

[1] https://python-poetry.org/; [2] https://docs.astral.sh/ruff/; [3] https://learn.microsoft.com/de-de/cli/azure/install-azure-cli

THE TEMPLATE – APP DEVELOPMENT & CONFIGURATION

Handling virtual environments across dev systems and the cloud

Project Configuration – pyproject.toml `pyproject.toml`

- Options for all aspects of a project and its environment
- Can use dynamic values and reference external files
- Can be parsed by python toml package
- Custom values are possible
  → used for pipeline parameters

```toml
[project]
# TODO: replace with actual project name, version, description, and author(s)
name = "TEC_002_Web_App_Template"  # Name must have this structure: [type in 3 letters: TEC/PAS/PRO]
version = "2.0.0"
description = "This is a template for Python projects that are hosted as an Azure App Service (i.e.
authors = [
  {name = "Johannes Schöck", email = "Johannes.Schoeck@nkd.de"},
  {name = "Luis Cuervo", email = "Luis.Cuervo@nkd.de"},
  {name = "Sai Nallamothu", email = "SaiRam.Nallamothu@nkd.de"},
]
readme = "README.md"
requires-python = ">=3.11"
dependencies = []

[tool.poetry]
package-mode = false
```

Virtual Environment – poetry

- Independent, project-specific virtual environment
- Needs python install (e.g., with anaconda)

[1] https://packaging.python.org/en/latest/guides/writing-pyproject-toml/; [2] https://python-poetry.org/