



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](https://github.com/JSchoeck/talk_webapp_template_and_pipeline_on_azure)

NKD





NKD GROUP

NKD Online Shop

<https://www.nkd.de/>

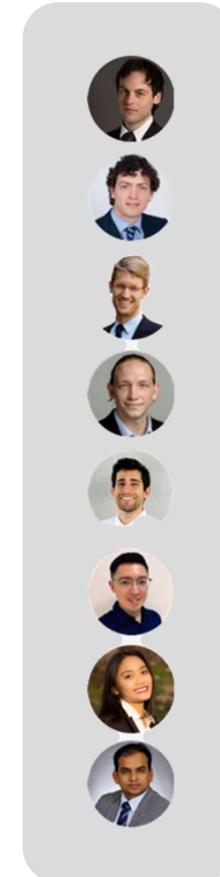
NKD Career

<https://www.nkdgroup.com/>

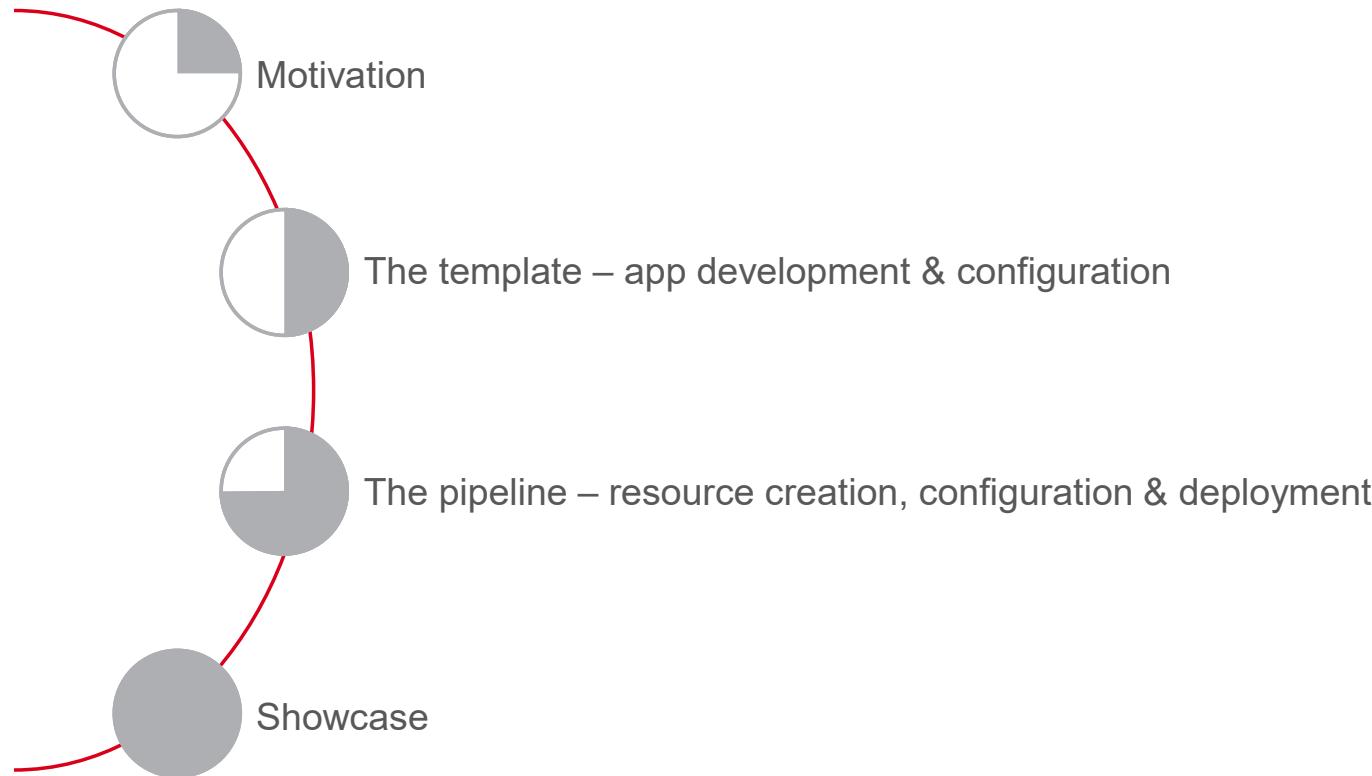
NKD Data Science Team

- 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



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

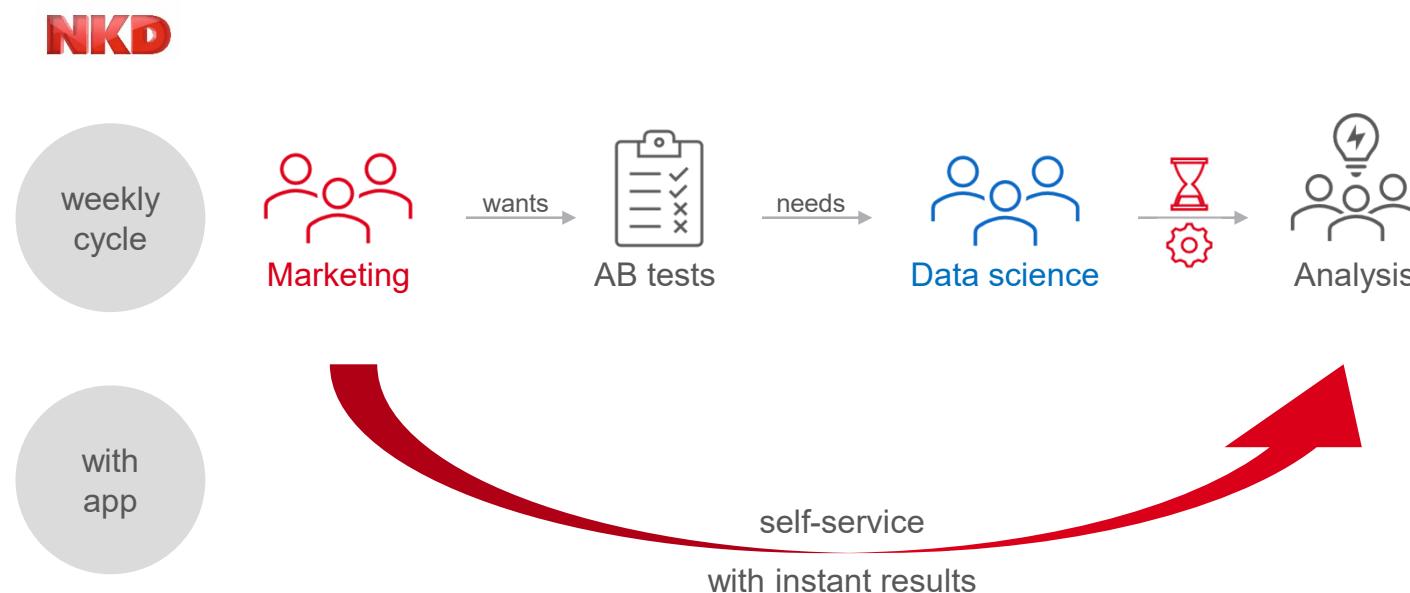
- Structure of the stages
- Minimizing manual configuration with file parsing and bicep
- Matching branch and target server
- Automated Azure resource creation using Azure CLI
- App authorization and authentication configuration with more Azure CLI
- Finally, the deployment

4. Show case

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

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



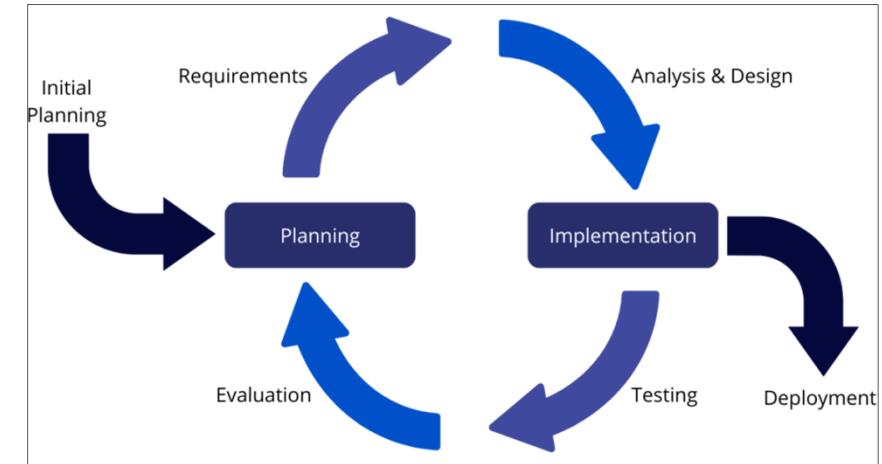
MOTIVATION

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

MOTIVATION

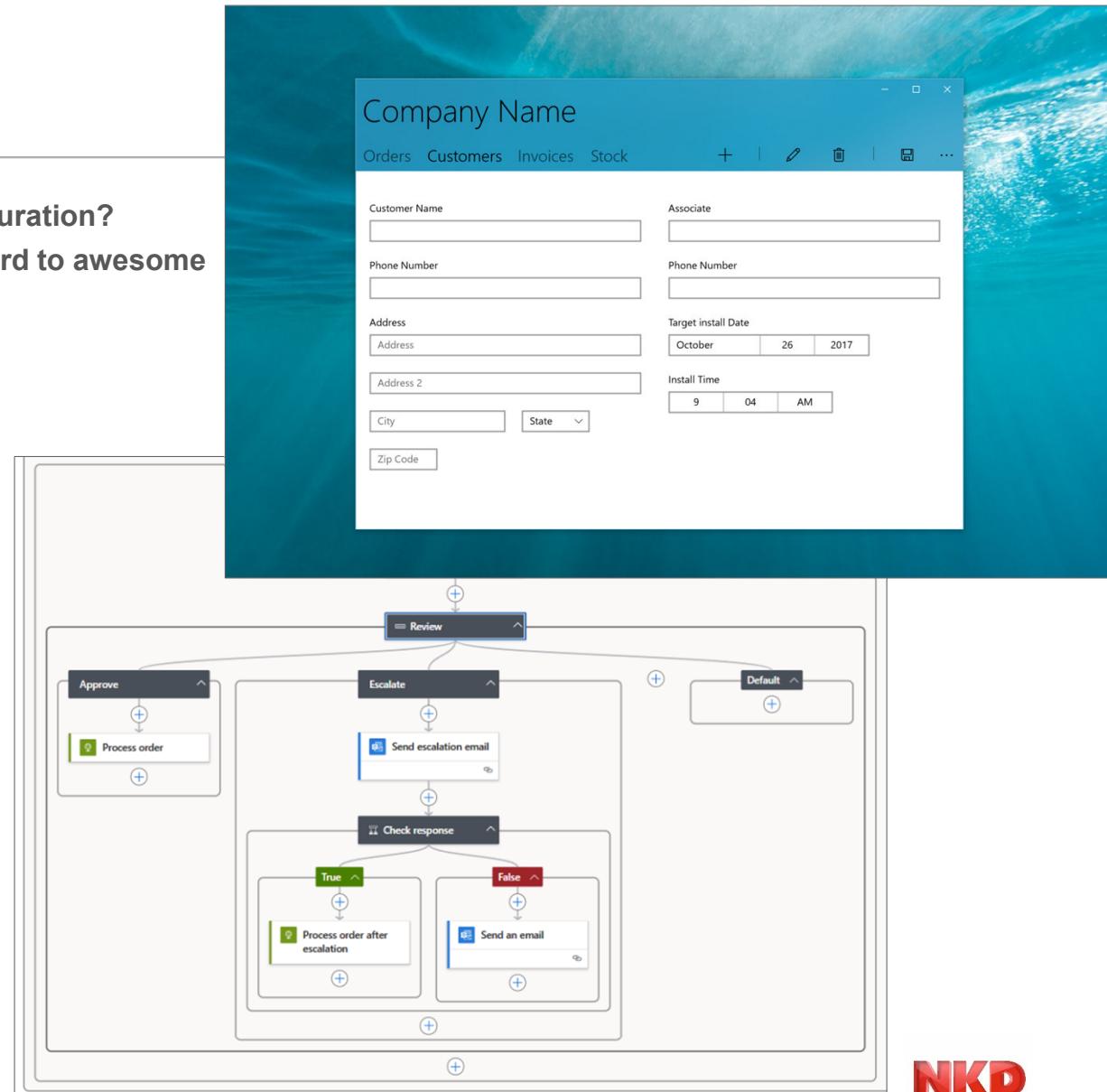
Why it's hard to get a real application running early

- With business users, discussing an app based on its code is usually not possible
→ UI required early during development
- 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
 - Additional cost / budget approval
 - Knowledge, effort & permissions to setup infrastructure
 - Safety concerns require user authentication & authorization

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 local Jupyter notebooks
 - Most apps run on request from business users
 - Interactive app using MS Forms and Logic apps
 - Not really file based → lack of version control
 - Hard to debug
 - Hard to parametrize & modularize
 - Hard to backup and setup after incident
 - 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
 - Shift from notebooks to using scripts
 - Common coding rules, using a linter and formatter
 - 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



CHALLENGES

- 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
- Virtual environment and dependencies vary across different systems
- Updates to the pipeline during project development leads to fragmented work
- When not actively working on the deployment pipeline, knowledge is lost again

Datenverarbeitung

Amazon EC2
Virtuelle Server in der Cloud

Amazon EC2 Auto Scaling
Rechenkapazität skalieren, um Anforderungen zu erfüllen

Amazon Lightsail
Virtual Private Server starten und verwalten

AWS App Runner
Erstellen und Ausführen von containerisierten Webanwendungen im großen Maßstab

AWS Batch
Ausführen von Batch-Jobs beliebiger Größe

AWS Elastic Beanstalk
Web-Apps ausführen und verwalten

AWS Lambda
Code ausführen, ohne an Server zu denken

Popular Azure services [See more](#)

 Function App Create	 Web App Create	 Logic App Create
 Azure AI Search Create	 App Service Plan Create	 Container App Create

Cloud Run

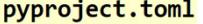
Cloud Run-Funktionen

App Engine

API Gateway

Endpunkte

TOOLS

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

More details in
this talk

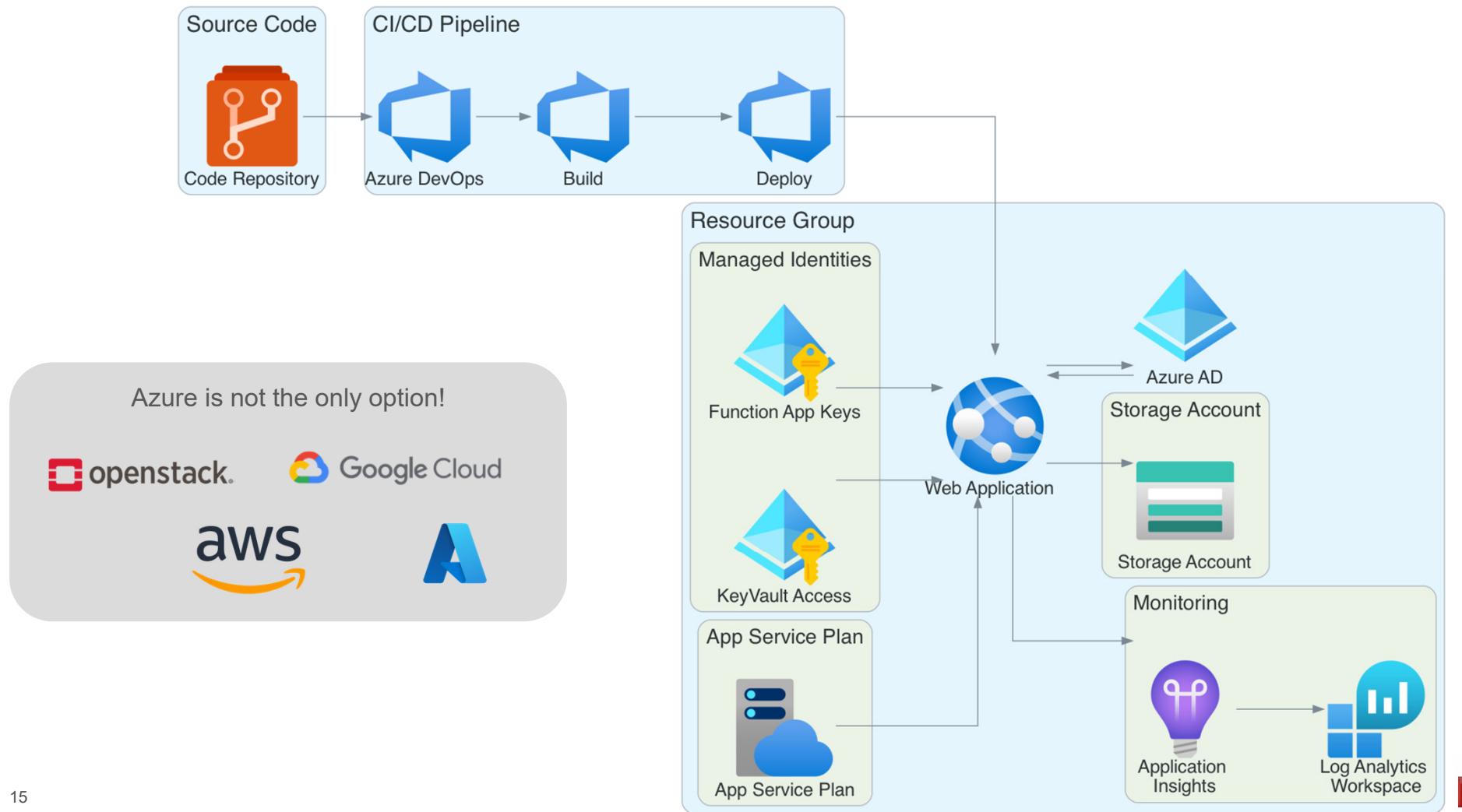


 https://github.com/JShoeck/talk_python_ruff-poetry
2024 @ Python User Group Nürnberg

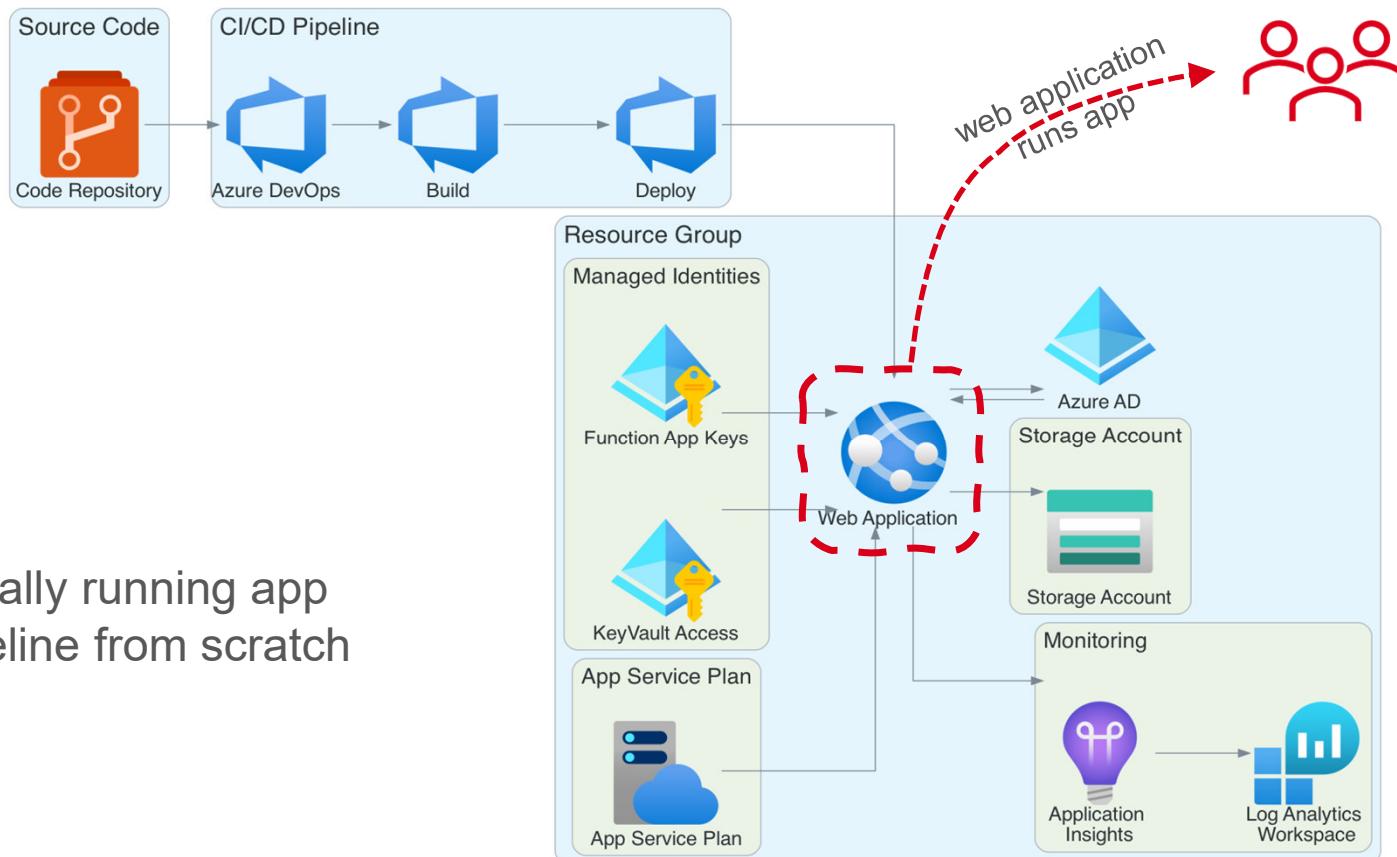
```
1 [project]
2 # TODO: replace with actual project name, version, description, and author(s)
3 name = "TEC_002_Web_App_Template" # Name must have this structure: [type in 3 letters: TEC/PAS/PRO]
4 version = "2.0.0"
5 description = "This is a template for Python projects that are hosted as an Azure App Service (i.e.
6 authors = [
7     {name = "Johannes Schöck", email = "Johannes.Schoeck@nkd.de"}, 
8     {name = "Luis Cuervo", email = "Luis.Cuervo@nkd.de"}, 
9     {name = "Sai Nallamothu", email = "SaiRam.Nallamothu@nkd.de"},
10 ]
11 readme = "README.md"
12 requires-python = ">=3.11"
13 dependencies = []
14
15 [tool.poetry]
16 package-mode = false
```



AZURE INFRASTRUCTURE



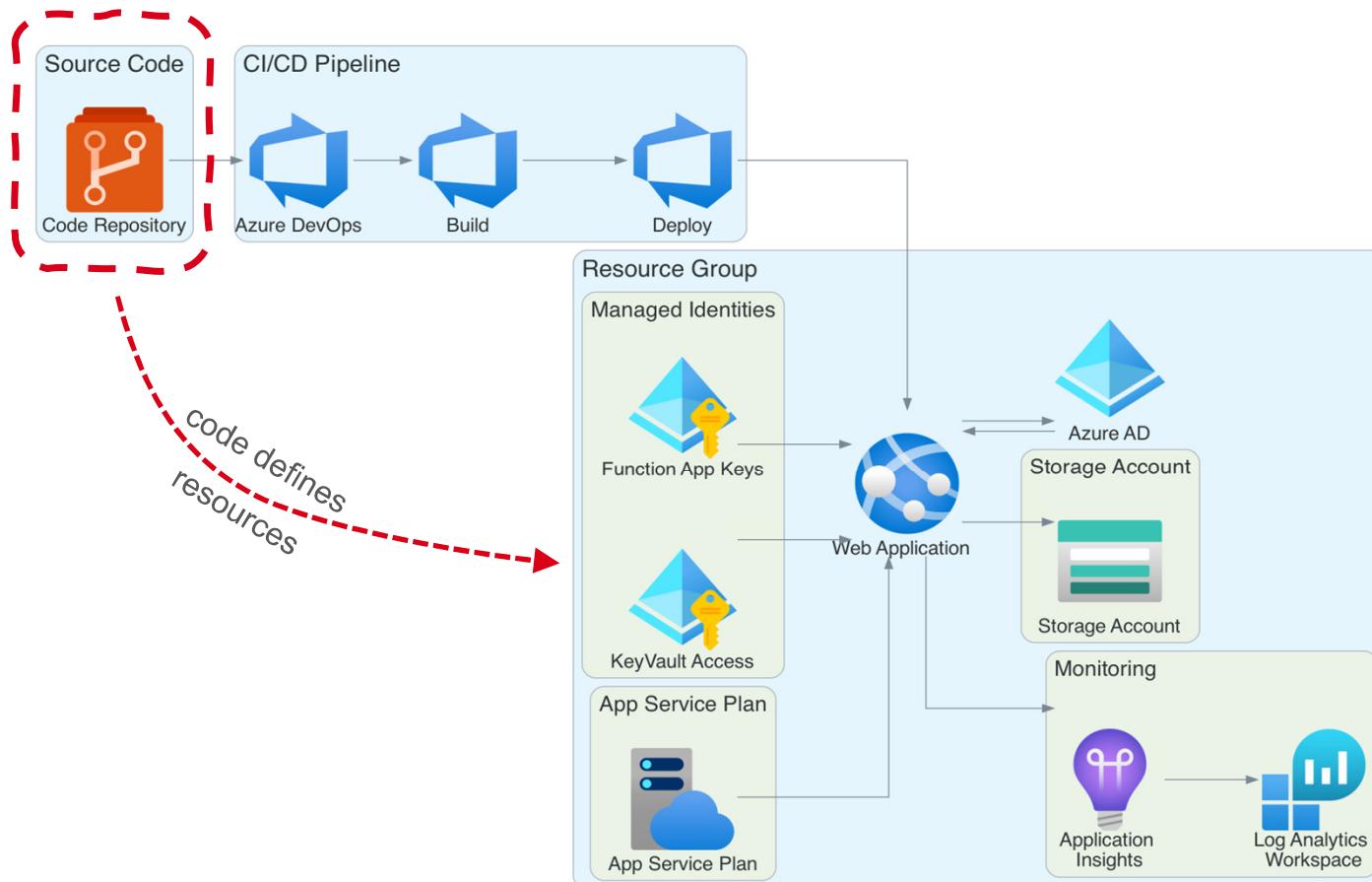
AZURE INFRASTRUCTURE – SHOWCASE



Live:

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

AZURE INFRASTRUCTURE





THE TEMPLATE – APP DEVELOPMENT & CONFIGURATION

- Poetry configuration in poetry.toml

```

1 [virtualenvs]
2 in-project = true
3
4 [virtualenvs.options]
5 always-copy = true

```

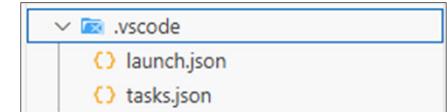
- Streamlit configuration in .streamlit\config.toml

```

1 [theme]
2 base="light"
3 primaryColor="#F63366"
4 backgroundColor="#FFFFFF"
5 secondaryBackgroundColor="#FADEDE" # production: F0F2F6; development: FADEDE
6 textColor="#262730"
7 font="sans-serif"
8
9 [client]
10 toolbarMode="minimal" # https://docs.streamlit.io/develop/concepts/architecture
11 showSidebarNavigation = false # Controls whether the default sidebar page
12 # Alternatively build custom navigation: https://docs.streamlit.io/develop
13
14 [browser]
15 gatherUsageStats = false # always set to false, otherwise user data will

```

- Running & debugging streamlit



```

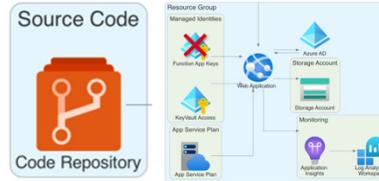
1 {
2   "version": "0.2.0",
3   "configurations": [
4     {
5       "name": "Streamlit app.py",
6       "type": "debugpy",
7       "request": "launch",
8       "module": "streamlit",
9       "env": {
10         "STREAMLIT_APP": "\\\src\\\app.py",
11         "PYTHONPATH": "${workspaceRoot}",
12         "PYDEVD_WARN_SLOW_RESOLVE_TIMEOUT": "30"
13       },
14       "args": [
15         "run",
16         "${workspaceRoot}\${STREAMLIT_APP}"
17       ],
18       "redirectOutput": true,
19       "preLaunchTask": "prepare environment",
20     },
21   ]
22 }

```

```

1 {
2   "label": "poetry install",
3   "type": "shell",
4   // Install dependencies from lock file into environment, without installing
5   "command": "poetry install --no-root",
6   "dependsOn": "poetry lock refresh"
7 }

```



THE TEMPLATE – DEPLOY.BICEP

Defines what resources get created

1. Fetch KeyVault ID, to avoid hardcoding it

```

1 resource keyVault 'Microsoft.KeyVault/vaults@2023-07-01' existing = {
2   scope: resourceGroup(subscriptionIdProd, utilitiesResourceGroupName )
3   name: 'kv-dataservice-general'
4
5 }
```

2. Create new resource group, with standardized name

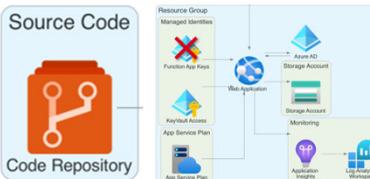
```

1 resource newRG 'Microsoft.Resources/resourceGroups@2022-09-01' = {
2   name: 'rg-${projectId}-${ projectName }'
3   location: location
4   tags: {
5     owner: 'data science'
6   }
7 }
```

3. Import resources.bicep with project parameters

```

1 module resources 'resources.bicep' = {
2   name: 'resources'
3   scope: newRG
4   params: {
5     projectId: projectId
6     projectName: projectName
7     location: location
8     utilitiesResourceGroupName: utilitiesResourceGroupName
9     pathToApp: pathToApp
10    pythonVersion: pythonVersion
11    runFromPackage: runFromPackage
12    blobStorage: blobStorage
13    skuProd: skuProd
14    skuDevtest: skuDevtest
15    existingASPname: existingASPname
16    miGeneralKeyvaultAccess: dsManagedIdentityKeyvaultAccessClientId
17    subscriptionIdProd: subscriptionIdProd
18  }
19 }
```



THE TEMPLATE – RESOURCES.BICEP

1. Define default parameters from deploy.bicepparam file
2. Set log analytics workspace to existing shared workspace
3. Set app service plan (ASP) to existing shared compute resource
4. Create storage account, blob service, and blob container and set their accessibility
5. Create application insights, targeting the log analytics workspace
6. Create ASP, if not using the shared one
7. Create web application
 - Assign managed ID to allow accessing KeyVault
 - 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

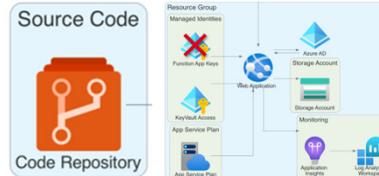


Streamlit configuration



- 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)

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



THE TEMPLATE – RESOURCES.BICEP

1. Define default parameters from deploy.bicepparam file
2. Set log analytics workspace to existing shared workspace
3. Set app service plan (ASP) to existing shared compute resource
4. Create storage account, blob service, and blob container and set their access
5. Create application insights, targeting the log analytics workspace
6. Create ASP, if not using the shared one
7. Create web application
 - Assign managed ID to allow accessing KeyVault
 - Set startup command for streamlit
 - Define environment variables to connect services and configure application settings
8. Create basic blocking identity provider

Resource interdependence

- Some resources depend on the existence of others
 - Creation order can be enforced using *dependsOn*
- ```
1 dependsOn: empty(existingASPname) ? [appServicePlan] : [existingAppServicePlan]
```
- Settings to allow resource access to other components are stored as application variables in the web app component
- ```
1 {
2   name: 'AZURE_STORAGE_ACCOUNT_NAME'
3   value: blobStorage ? storageAccount.name : ''
4 }
```

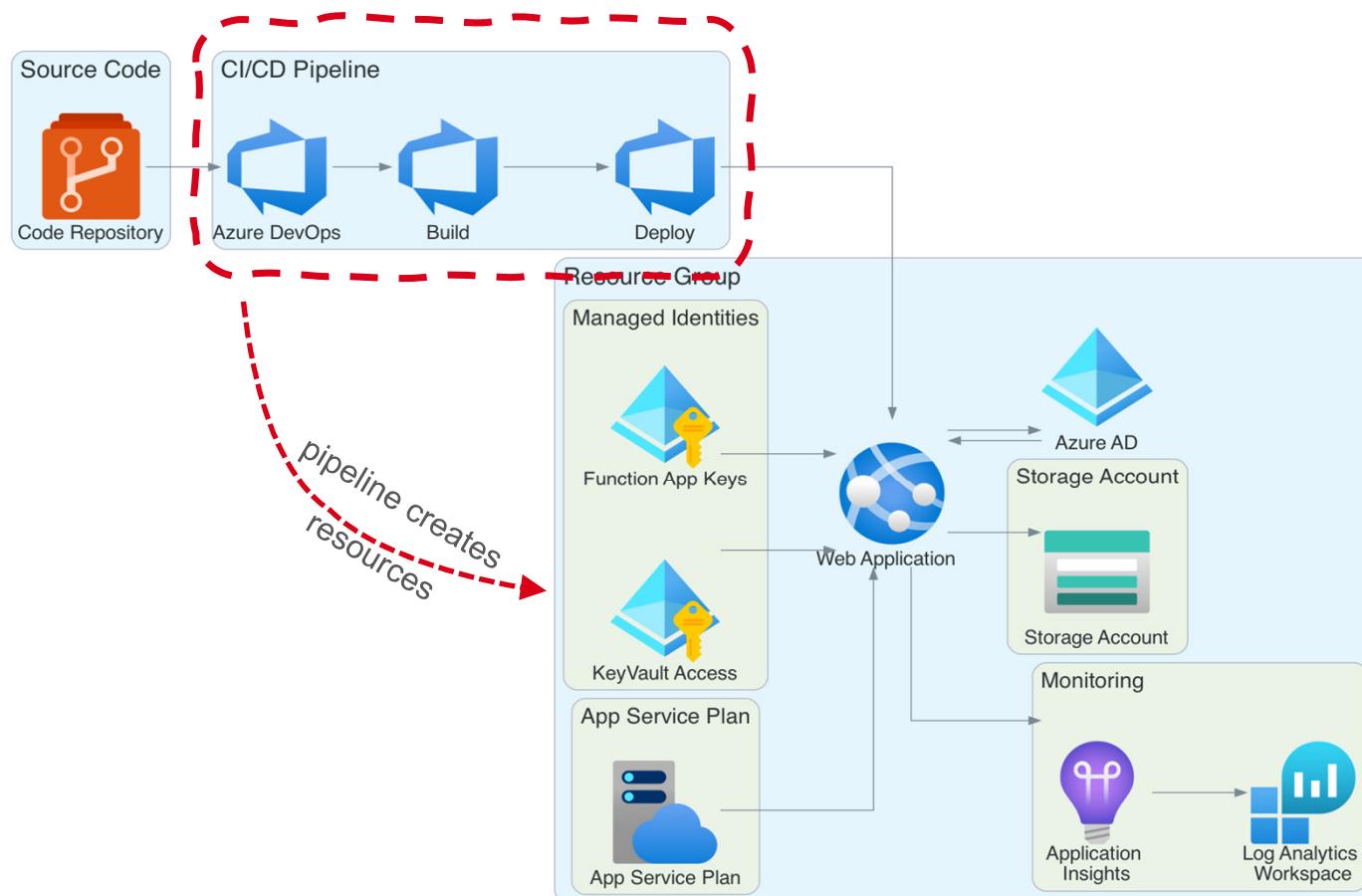


Naming

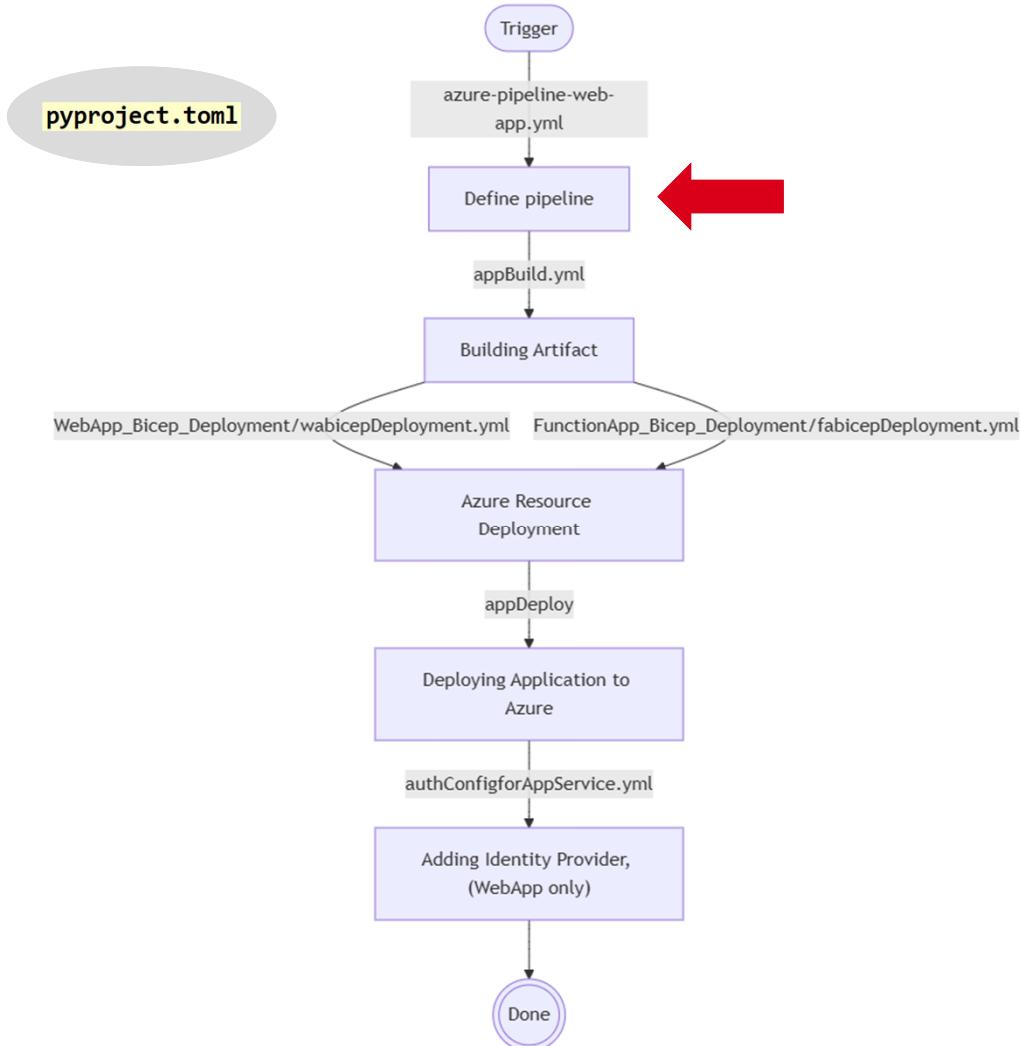
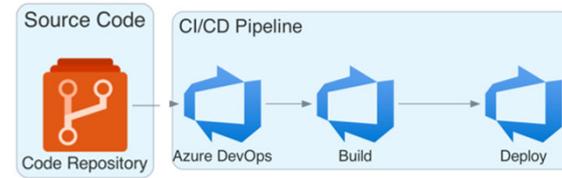
- Resources have fixed naming schemes
- Split into production and development environment is represented in most resource names by adding “-dev”

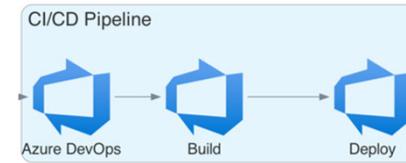
```
1 resource webApplication 'Microsoft.Web/sites@2024-04-01' = {
2   name: environment == 'production' ? 'wa-${projectFullName}' : 'wa-${projectFullName}-dev'
```

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

Variables

Name ↑	Value
armConnectionDev	connection-to-devtest-subscription

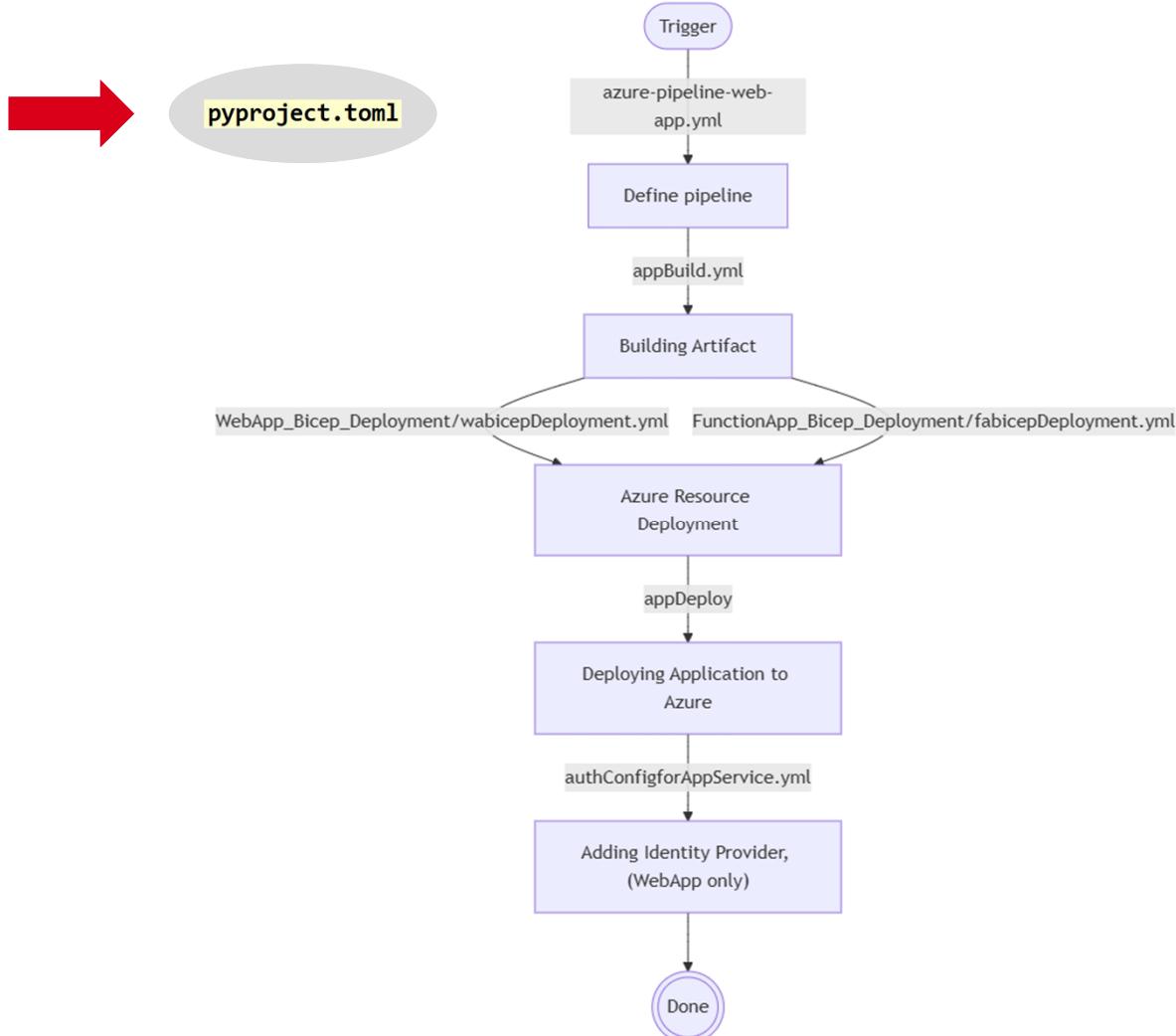
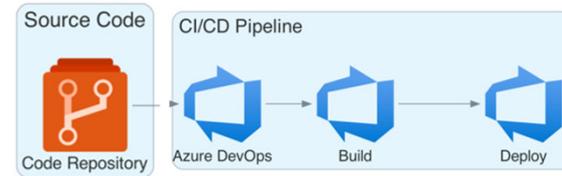
→ Pipeline variables for common settings used only in the pipeline

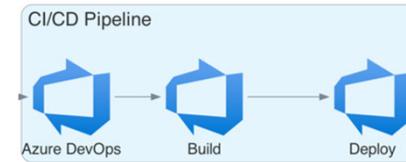
```
1 variables:
2   - name: isMain
3     value: $[eq(variables['Build.SourceBranchName'], 'main')]
```

→ Matching branch and target server

```
stages:
- template: appBuild.yml@Shared_Templates
- template: WebApp_Bicep_Deployment/wabicepDeployment.yml@Shared_Templates
  parameters:
    projectId: $[stageDependencies.Build.BuildJob.outputs['setVars.projectId']]
```

THE PIPELINE – STRUCTURE OF THE STAGES





THE PIPELINE – PARAMETERS

Minimizing manual configuration with config file parsing

- Parse parameters from our one and only source of adaptable configuration: `pyproject.toml`
- Easy to implement error-handling and variable modifications
- Set parameters as environment variables in the pipeline

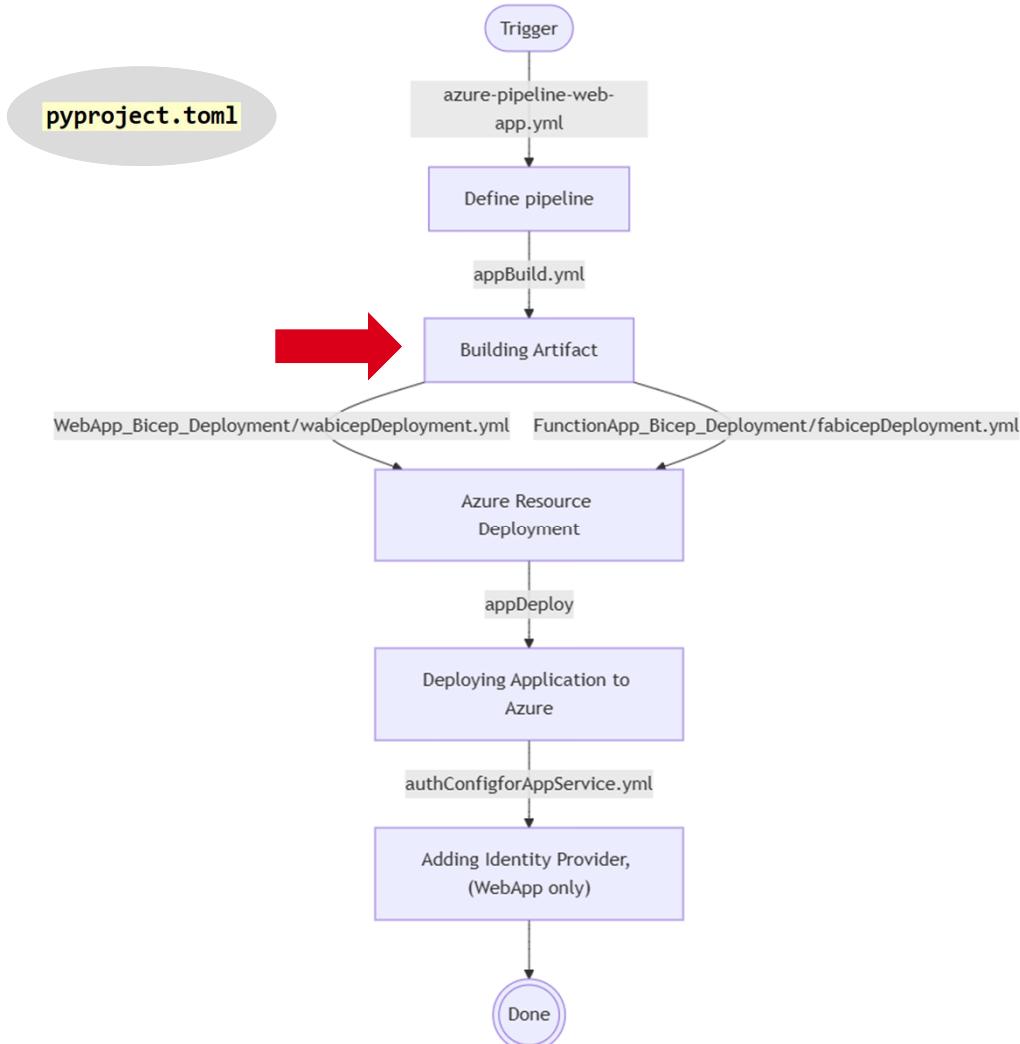
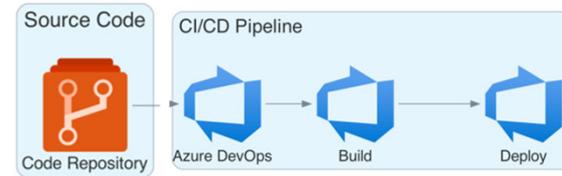
`ado_pipeline_repo\WebApp_Bicep_Deployment\deploy\nkd_pipeline_utils.py`

```

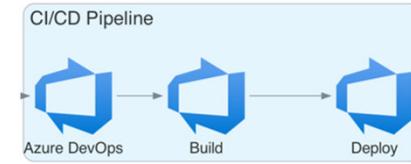
1 def get_var_from_pyproject(var_name: str, path: Path | None = None) -> None:
2     if path is None:
3         path = Path.cwd().joinpath("pyproject.toml")
4     config = toml.load(path)
5     if var_name == "pythonVersion":
6         var_value = clean_version_string(config["project"]["requires-python"])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

```

THE PIPELINE – STRUCTURE OF THE STAGES



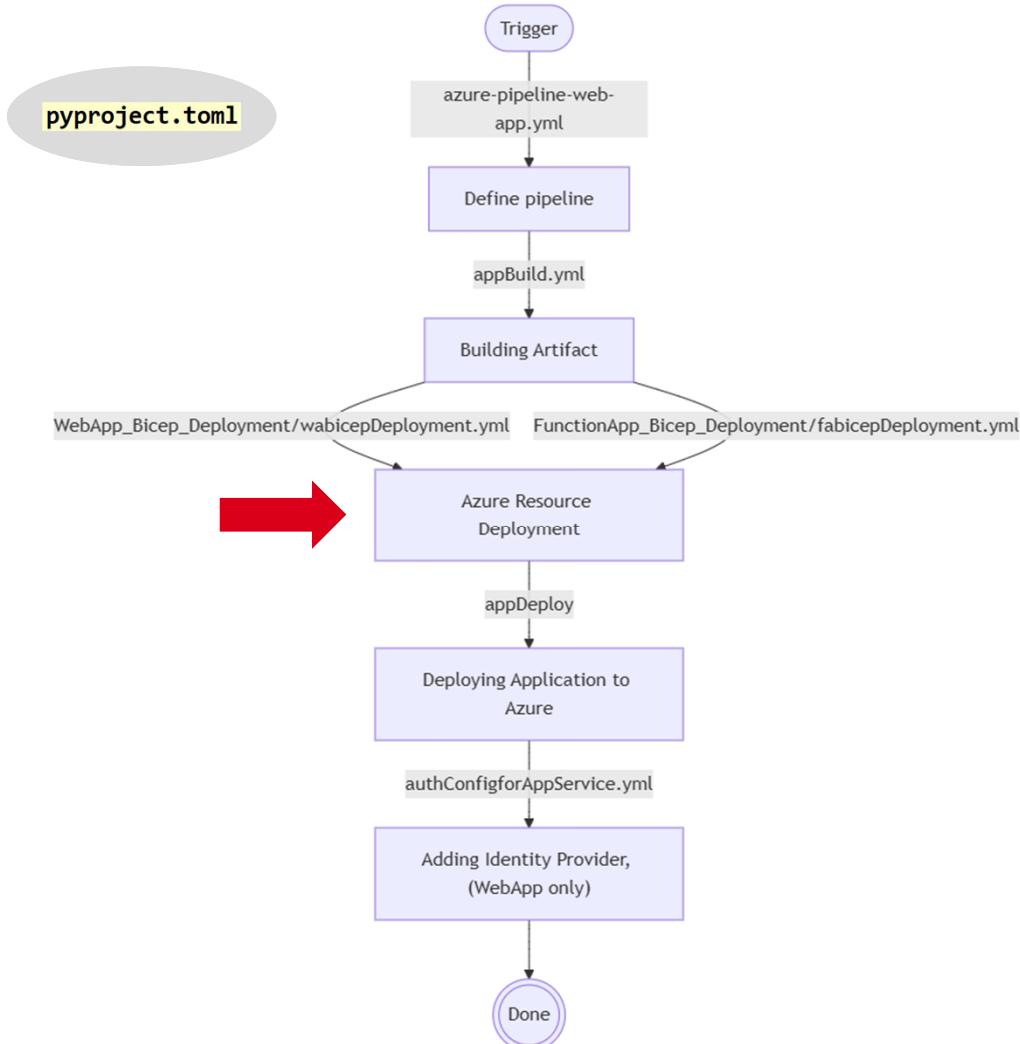
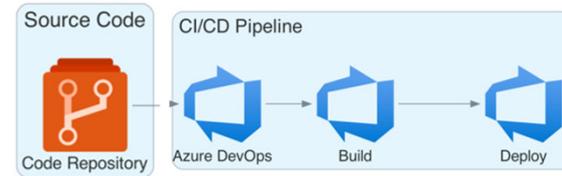
THE PIPELINE – APPBUILD.YML

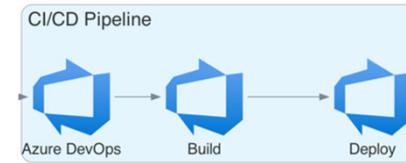


Steps of build stage

1. Find pyproject.toml and set path as variable
 2. Extract project variables to set them globally
 3. Process pipeline template and 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 venv and web app files
 10. Upload zip file from build to pipeline server
- Export to requirements.txt and pip install from there to handle missing native support for poetry
 - Install requirements into target environment's *site-packages* folder
 - Delete files that are not required to run the web app to decrease deployed file size and avoid accidental publication of secrets

THE PIPELINE – STRUCTURE OF THE STAGES





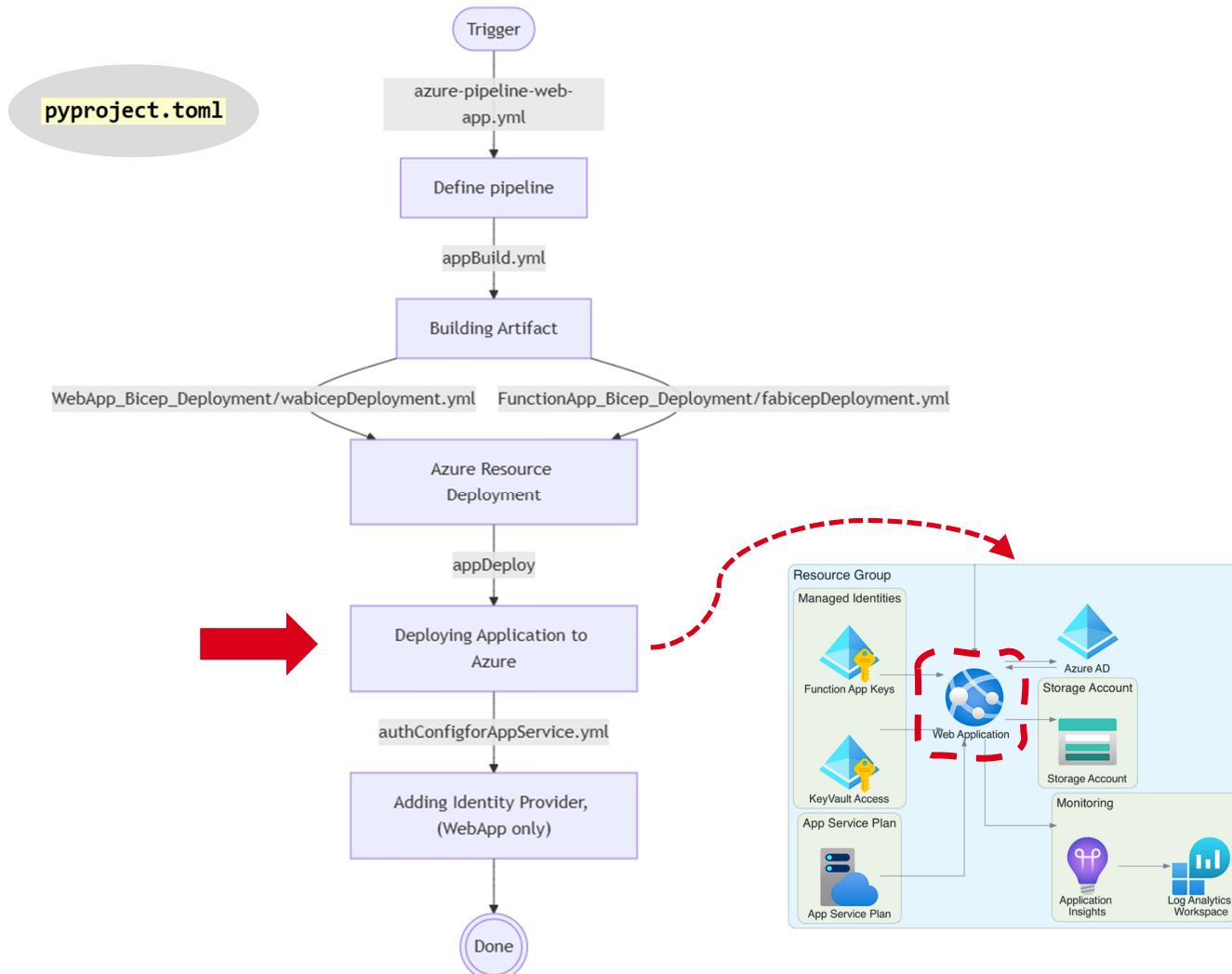
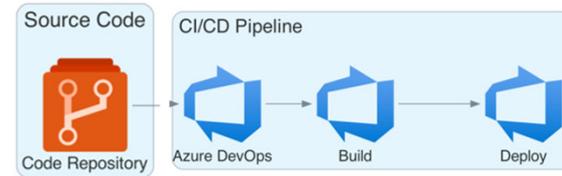
THE PIPELINE – WABICEPDEPLOYMENT.YML

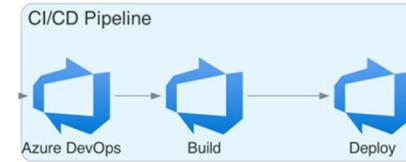
Steps of Azure resource creation stage

1. Define pipeline variables
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

- All parameters are received from calling pipeline
- Stage only runs on success of previous build stage
- Resource definition happens in .bicep file
- Default settings for resources are defined in .bicepparam file
- Azure CLI is used in pipeline bash scripts

THE PIPELINE – STRUCTURE OF THE STAGES





THE PIPELINE – APPDEPLOY.YML

Steps of deployment stage

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

- All parameters are received from calling pipeline
- Stage only runs on success of all previous stages

```

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

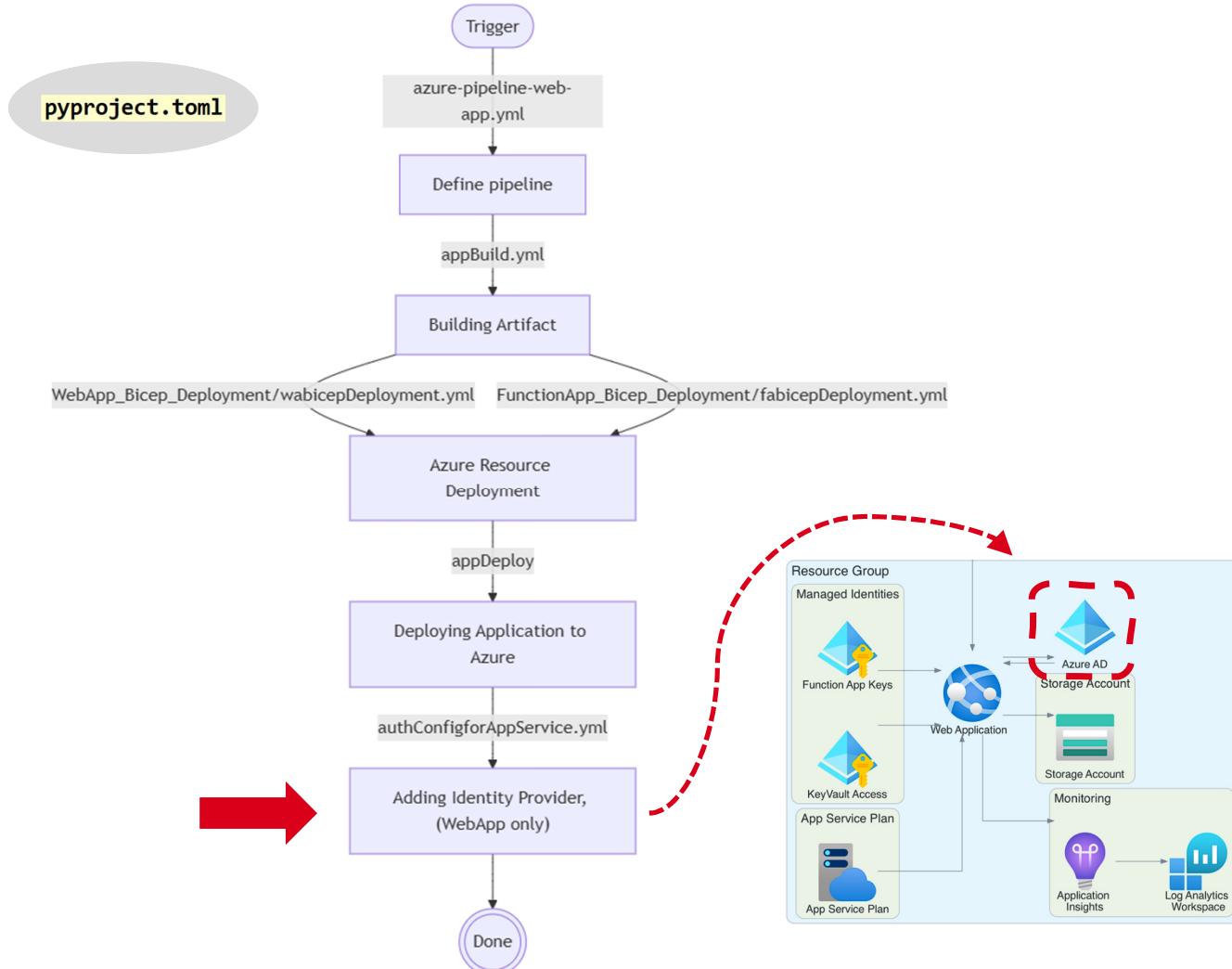
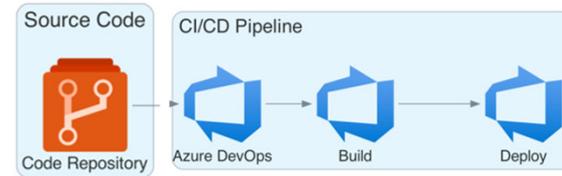
```

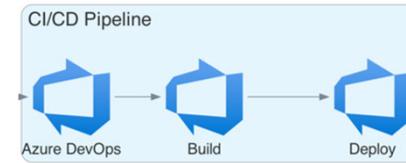
```

1 parameters:
2   - name: armconnection
3     type: string
4   - name: appName
5     type: string
6   - name: appType
7     type: string
8   - name: projectId
9     type: string
10  - name: projectName
11    type: string
12
13 stages:
14 - stage: DeployApp
15   displayName: 'Deploy to azure environment'
16   dependsOn:
17     - Build
18     - DeployBicep
19   variables:
20     armconnection: ${{ parameters.armconnection }}
21     appName: ${{ parameters.appName }}
22     appType: ${{ parameters.appType }}
23     isMain: ${{ variables.isMain }}
24     projectId: ${{ parameters.projectId }}
25     projectName: ${{ parameters.projectName }}

```

THE PIPELINE – STRUCTURE OF THE STAGES



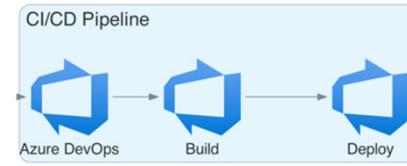


THE PIPELINE – AUTHCONFIGFORAPPSERVICE.YML

Steps of authorization configuration stage

1. Define pipeline variables
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

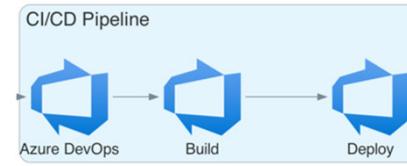
- All parameters are received from calling pipeline
- Stage only runs on success of all previous stages



THE PIPELINE – CREATION IN ADO

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

The screenshot illustrates the Azure DevOps Pipeline creation interface. On the left, the sidebar shows the 'Data Science' project. The main area is divided into four tabs: Connect, Select, Configure, and Review. The 'Connect' tab is active, showing options for connecting to code repositories like Azure Repos Git, Bitbucket Cloud, GitHub, and GitHub Enterprise Server. A red arrow points to the 'Azure Repos Git' option, which is highlighted. The 'Select' tab shows a list of repositories, with one named 'PAS_000_demo_app' selected, indicated by a red arrow. The 'Configure' tab is the active tab, titled 'Configure your pipeline'. It lists several pipeline types: Python package, Python to Linux Web App on Azure, Starter pipeline, and Existing Azure Pipelines YAML file. A red arrow points to the 'Existing Azure Pipelines YAML file' option. A modal window titled 'Select an existing YAML file' is open, showing a dropdown menu for selecting a branch ('main') and a path input field containing '/azure-pipeline-web-app.yml'. Another red arrow points to this modal window.



THE PIPELINE – CREATION IN ADO

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

Azure DevOps nkdit / Data Science / Pipelines / PAS_000_demo_app / 20250630.1

BuildJob

This pipeline needs permission to access a resource before this run can continue to Build stage

1 Job is pending...

Checks and manual validations for Build stage

Permission Repository TEC_007_Shared_Templates Permission needed Permit

Checks and manual validations for Deploy to azure environment

Permission Environment development Permission needed Permit

Stages

Build stage: 1 job completed, 4m 38s, 1 artifact

Deploy Bicep Template: 1 job completed, 2m 22s

Deploy to azure envir...: Waiting

Permission needed

NKD

THE PIPELINE – DEBUGGING THE DEPLOYMENT

- Debugging deployment errors:

Azure Monitor – Activity Log

The screenshot shows the Microsoft Azure Monitor Activity log page. The left sidebar lists various monitoring features: Overview, Activity log (which is selected), Alerts, Metrics, Logs, Change Analysis, Service health, Workbooks, Dashboards with Grafana (preview), Insights (Applications, Virtual Machines, Storage accounts, Containers, Networks, Azure Cosmos DB, Key Vaults, Azure Cache for Redis, Azure Data Explorer Clusters), and Metrics. The main pane displays a table of activity logs with the following columns: Activity, Edit columns, Refresh, Export, and a list of log entries. The log entries include:

- Create Deployment
- Create Deployment
- Update insights component
- Update Storage Account Create
- 'audit' Policy action.
- Update insights component
- 'auditIfNotExists' Policy action.
- Update Storage Account Create
- Update Storage Account Create
- Get deployment operation status
- List Storage Account Keys
- List Storage Account Keys
- Put blob service properties
- Put blob service properties
- Put blob container
- Get deployment operation status
- Put blob container
- Update website
- Update website
- Get deployment operation status

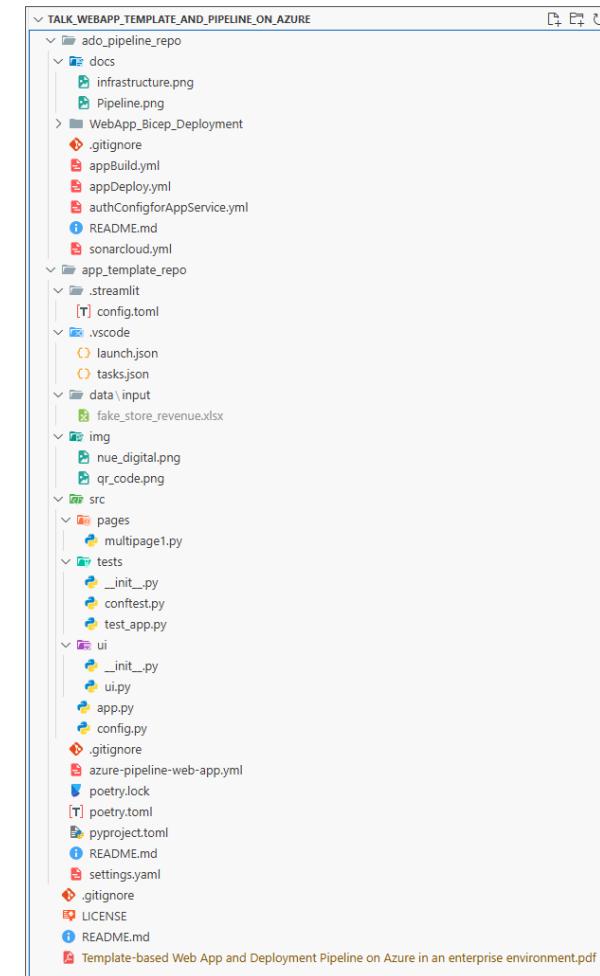
A red circle with a white exclamation mark is overlaid on the bottom right corner of the log table.

SHOWCASE

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

```
✓ Deploy Azure Web App

1 Starting: Deploy Azure Web App
2 =====
3 Task      : Azure Web App
4 Description : Deploy an Azure Web App for Linux or Windows
5 Version    : 1.257.0
6 Author     : Microsoft Corporation
7 Help       : https://aka.ms/azurewebapptroubleshooting
8 =====
9 Got service connection details for Azure App Service:'wa-PAS-000-demo-app'
10 Package deployment using ZIP Deploy initiated.
11 Deploy logs can be viewed at https://wa-pas-000-demo-app.scm.azurewebsites.net/api/deployments/4ec51e29-7f6c-49b8-a394-d80cdd1f15dc/log
12 Successfully deployed web package to App Service.
13 Successfully updated App Service configuration details
14 Successfully added release annotation to the Application Insight : wa-PAS-000-demo-app-appinsights
15 Successfully updated deployment History at https://wa-pas-000-demo-app.scm.azurewebsites.net/api/deployments/231381751290632708
16 App Service Application URL: https://wa-pas-000-demo-app.azurewebsites.net
17 Finishing: Deploy Azure Web App
```



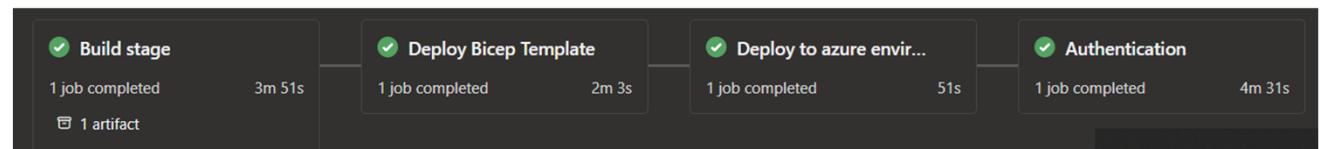
YOU WANT MORE?

- 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
- 1 `` `bash
2 curl -u :<PAT> \
3 -X POST \
4 -H "Content-Type: application/json" \
5 -d '{
6 "folder": "\\",
7 "name": "MyPipeline",
8 "configuration": {
9 "type": "yaml",
10 "path": "azure-pipelines.yml"
11 }
12 }' \
13 https://dev.azure.com/{organization}/{project}/_apis/pipelines?api-version=6.0-preview.1
14`
- 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

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:
super fast automated deployment!





QUESTIONS?

COMMENTS?

SUGGESTIONS?

Code and slides @GitHub



 [https://github.com/JSchoeck/
talk_webapp_template_and_pipeline_on_azure](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

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

```
1 [project]
2 # TODO: replace with actual project name, version, description, and author(s)
3 name = "TEC_002_Web_App_Template" # Name must have this structure: [type in 3 letters: TEC/PAS/PRO]
4 version = "2.0.0"
5 description = "This is a template for Python projects that are hosted as an Azure App Service (i.e.
6 authors = [
7     {name = "Johannes Schöck", email = "Johannes.Schoeck@nkd.de"}, 
8     {name = "Luis Cuervo", email = "Luis.Cuervo@nkd.de"}, 
9     {name = "Sai Nallamothu", email = "SaiRam.Nallamothu@nkd.de"}]
10 ]
11 readme = "README.md"
12 requires-python = ">=3.11"
13 dependencies = []
14
15 [tool.poetry]
16 package-mode = false
```



Virtual Environment – poetry

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