Integration Documentation

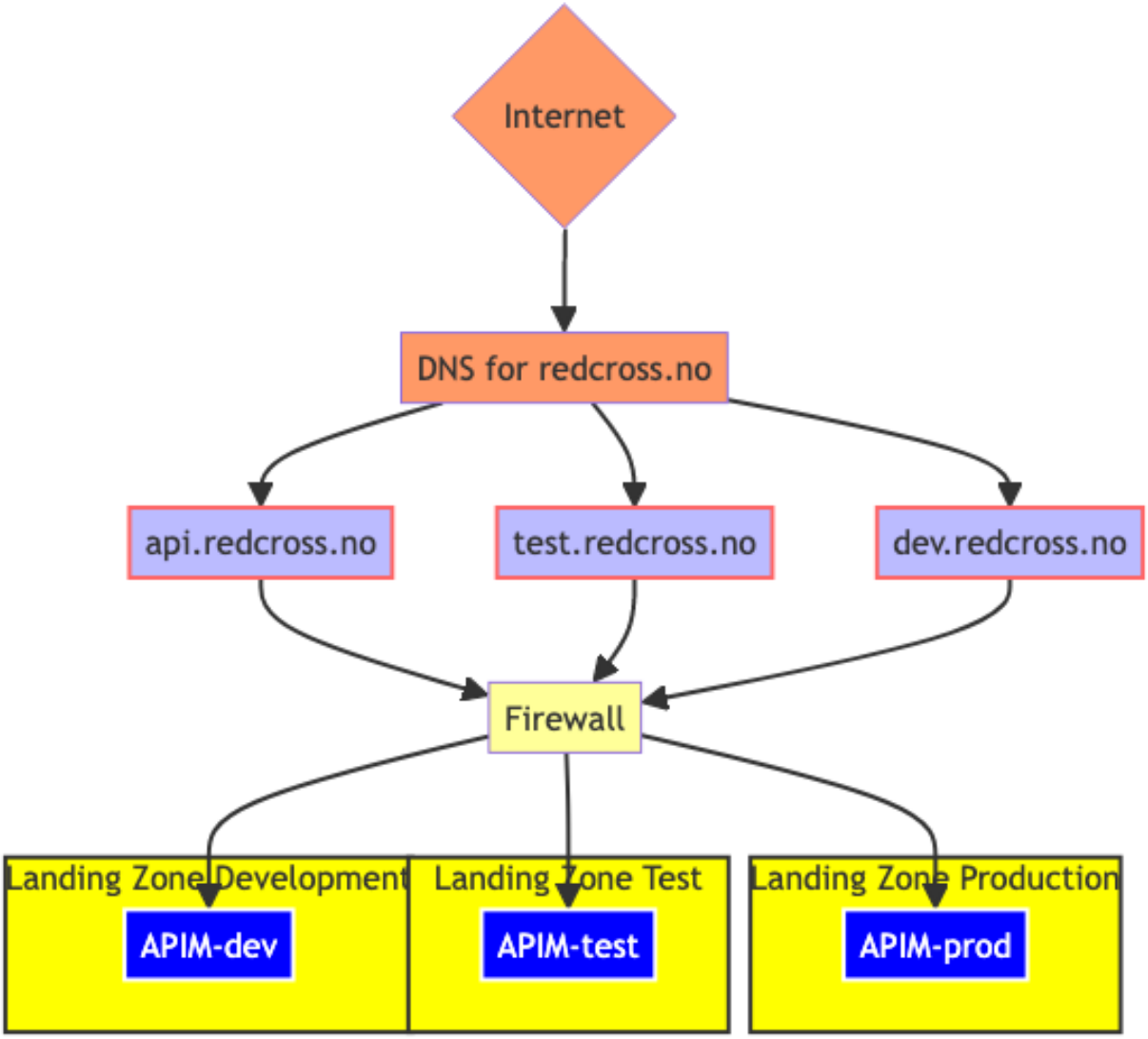
Generated from https://dev.azure.com/RedCrossNorway/Integrations

Warning: This PDF document is generated from the markdown files in the repository. Consider it outdated if not generated recently. Generated at Fri Apr 26 21:32:30 CEST 2024

Contents

External Context

The external facing part of the network is the part that is visible to the public.



Subdomains and their routing

Our domain is redcross.no The new infrastructure will have 3 subdomains that are exposed to the outside. These are:

Subdomain	Description	Dest landing zone	Dest “host”
api.redcross.no	Production API	Landing Zone Production	APIM-prod
test.redcross.no	Test API	Landing Zone Test	APIM-test
dev.redcross.no	Development API	Landing Zone Development	APIM-dev

All of these subdomains are pointing to the same firewall.

The firewall routes traffic based on the subdomain to the correct landing zone.

DNS

redcross.no is the domain. It is hosted on the DNS server in the “Betala per anvending” subscription.

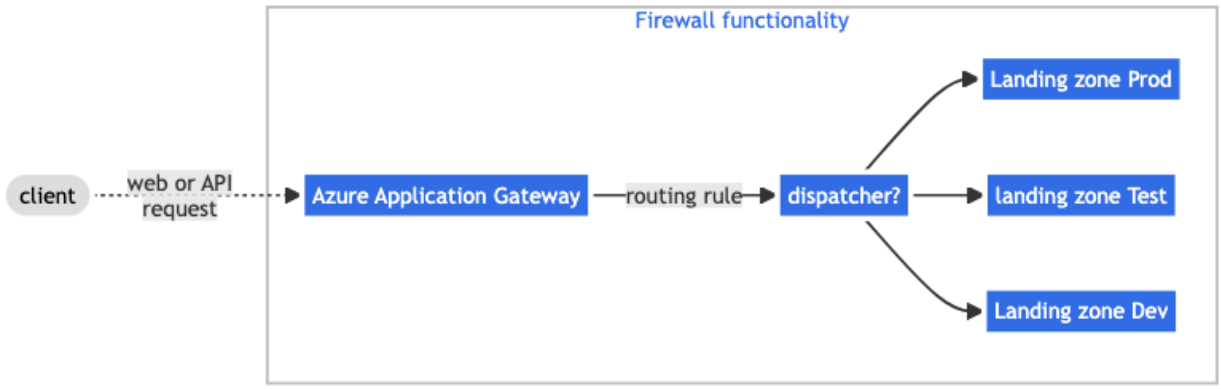
When we set up the new subscription we point the DNS records defined in External facing to the new DNS server.

Firewall

The firewall is the first line of defense for the network. It is the only part of the network that is exposed to the internet. The firewall routes traffic based on the subdomain to the correct landing zone.

Firewall functionality

We are using Azure Application Gateway as a firewall. The Application Gateway is the only resource with a public IP address. The Application Gateway terminates SSL and routes traffic based on subdomain.



For mre details see:

What	Description
Firewall Rules	describes the firewall rules
Firewall SSL termination	describes the SSL termination on the firewall
Firewall doc	describes firewall doc
Firewall logging	describes how to set up logging for the firewall

Firewall documentation

We are using Azure Application Gateway as a firewall. Documentation for the product is available at Azure Application Gateway documentation.

Our special configuration is described below.

TODO: Describe the configuration of the firewall.

Firewall Rules

Below are the firewall rules for integrations in Red Cross Norway.

hostname	Source	Destination	Port	Protocol	landing Zone	Description
api.red-cross.no	*	APIM-prod	443	HTTPS	Production	API for the Red Cross
api.red-cross.no	*	APIM-prod	80	HTTP	Production	API for the Red Cross
test.red-cross.no	*	APIM-test	443	HTTPS	Test	test API for the Red Cross
test.red-cross.no	*	APIM-test	80	HTTP	Test	test API for the Red Cross
dev.red-cross.no	RC Intranet	APIM-dev	443	HTTPS	Development	dev API for the Red Cross
dev.red-cross.no	RC Intranet	APIM-dev	80	HTTP	Development	dev API for the Red Cross

NOTE: The `RC Intranet` is the internal network for the Red Cross Norway. It is not accessable from the internet.

Firewall SSL termination

Our SSL certificate is a wildcard certificate for redcross.no. The certificate is installed on the Azure Application Gateway.

Currently the certificate is installed on the Firewall on the “Betala per anvending” subscription. When we set up the new subscription we will install the certificate on the Azure Application Gateway. Then we will point the DNS records defined in External facing to the new Application Gateway.

TODO: Jah - is the certificate installed on the FW in the new subscription or the old?

Firewall Logging

This document describes how to set up logging for the firewall. We are collecting all logs in Sentinel.

TODO: Add information about logging and who is responsible for monitoring the logs.

Landing Zones

Integrations are using three landing zones. The landing zones are separate environments for production, test, and development. The landing zones are isolated from each other. Each landing zone has its own set of resources.

Integration Landing Zones

The landing zones are:

Landing Zone	Long name	Description
landing-prod	prod - azure integrations -az - red cross	production landing zone
landing-test	test - azure integrations -az - red cross	test landing zone
landing-dev	dev - azure integrations -az - red cross	development landing zone

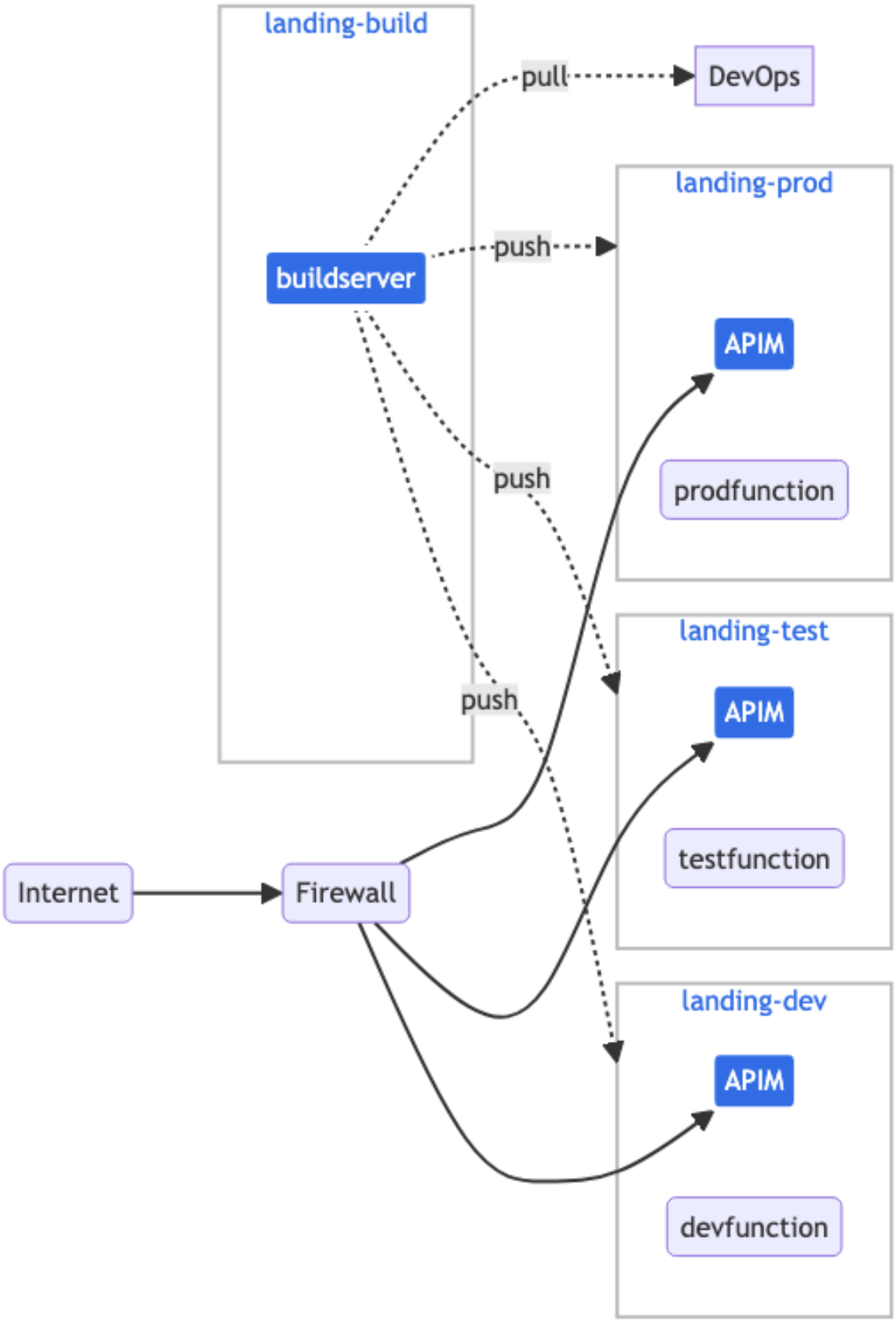
IMPORTANT: Isolation The landing zones are isolated from each other. It means that an application running in one landing zone cannot access a resource in another landing zone. The landing zones are isolated to prevent unauthorized access to resources.

IMPORTANT: No inbound access There is no way to access any resources in the landing zones from the internet or the redcross internal network. All access to the landing zones is through the firewall.

Because of the isolation, the build server is the only way to deploy applications to the landing zones. The build server has access to all landing zones and can deploy applications to any of them. See more about the build server in the build server documentation.

Landing Zones diagram

The diagram below shows the landing zones and the resources in each landing zone.


landing-build
pull
DevOps
landing-prod
push
buildserver
APIM
push
prodfunction
landing-test
push
APIM
testfunction
Internet
Firewall
landing-dev
APIM
devfunction

Landing Zone documentation overview

Landing zones are like subnets. They are isolated from each other. As a general rule there are no comminication between landing zones.

There can be exceptions to this rule, but they should be well documented.

Landing Zone Production

The production landing zone is where the production systems are located. The landing zone is accessable from the internet.

Landing Zone Test

This zone is accessable from the internet. It is used for testing systems before they are moved to production. It has the same security rules as the production zone.

Landing Zone Development

This zone is used for development. It is not accessable from the internet.

TODO: The dev landing zone can only be accessed from the internal network. How is this done?

Landing Zone Build

This zone is used for the build server. It is not accessable from the internet.

Landing Zone Production details

The production landing zone is where the production systems are located. The landing zone is accessable from the internet.

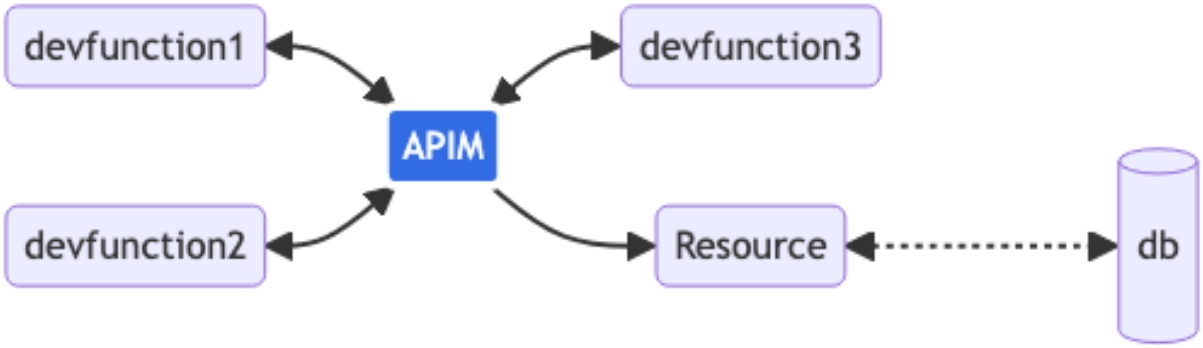
Any details about the production landing zone should be documented here.

Communication rules

All communication between systems must follow these rules.

Internal communication (between systems internally)

Communication between internal systems must use Azure API Management (APIM). This is to ensure that we have a single point of entry for all communication. This makes it easier to monitor and secure the communication.

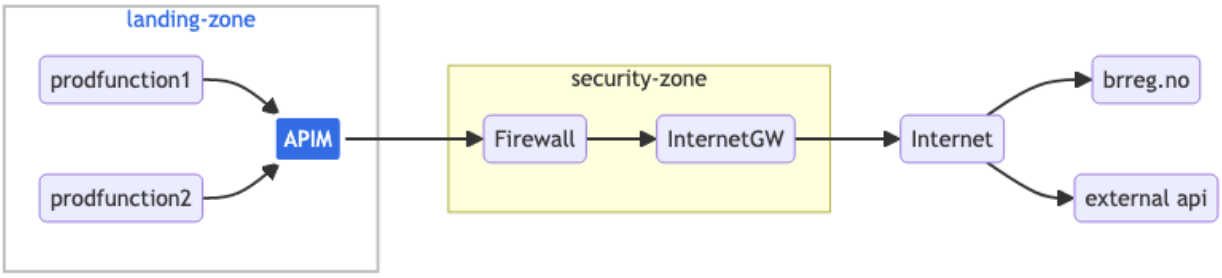


In the example above, the `devfunction` is an Azure Function that is calling a `resource`. The resource is a web service that is accessing a database. The Azure Function is calling the resource through APIM.

Functions can only call resources through APIM.

External communication (from internal systems to external systems)

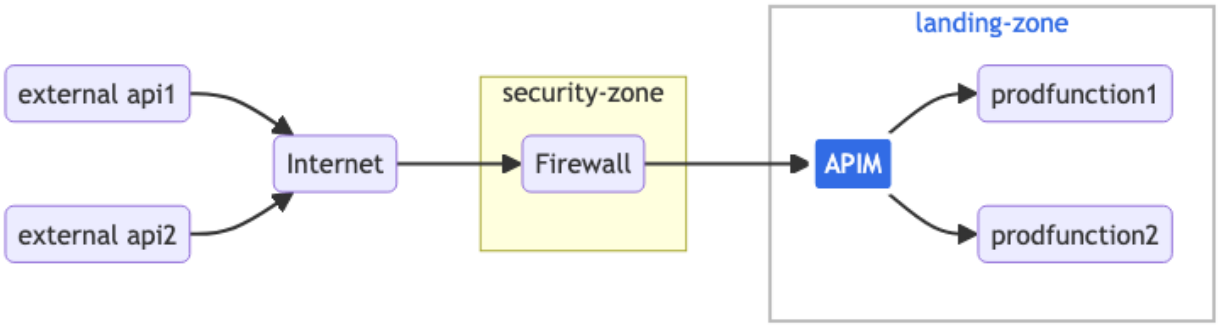
Communication from internal systems to external systems must use Azure API Management (APIM). We are doing this so that we can respond to changes on external APIs. If an external API changes, we can update the APIM to reflect the changes. This way, we don't have to update all the internal systems that are calling the external API.



Requests from external systems to internal systems

Requests from external systems to internal systems will first go trugh the firewall and then to APIm in the landing zone. See External facing for more information.

APIM will route the request to the correct internal system.



Definition of external systems

All systems on the old Azure "Betala per andvending" are external systems.

External systems are systems that are not part of the new Azure environment. These systems are accessed through the internet.

TODO: we need to make everyone aware of the consequences of this.

Communication rules exceptions

This document describes what is needed to make an exception to the communication rules defined in 6-0communication-rules.md.

TODO: If there should ever be exceptions to the comminication rules. We ned to document how we make exceptions to the rules.

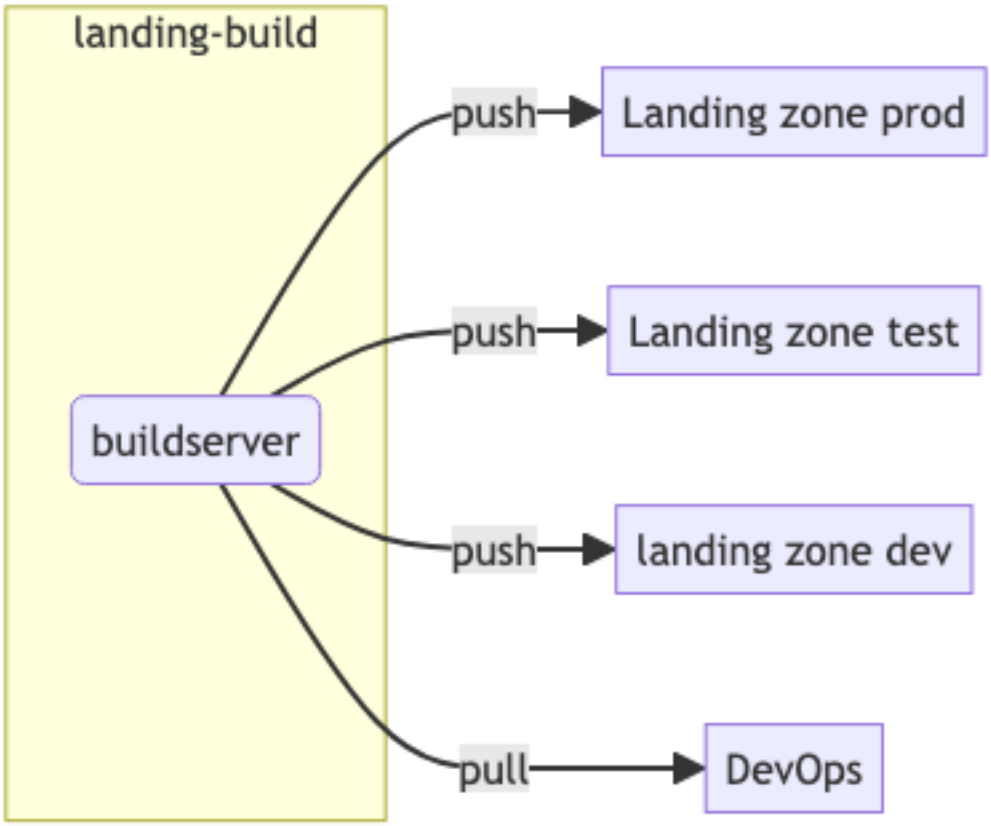
Development

This document describes the development setup for Red Cross Norway. It is intended for developers who create and maintan integrations.

- Build server
- DevOps
- Azure Functions
- Naming conventions
- Infrastructure as code
- Development Security

Build server

Deploy from DevOps does not work because DevOps has no access into the Landing zones. There is therefore a build server.



The build server is a virtual machine that has access to the landing zones. The build server is monitoring the source code repository for changes. When a change is detected, the build server pulls the source code and builds the application. The build server then deploys the application to the correct landing zone.

How to set up the build server for a repository

TODO: Describe how to set up the build server for a repository.

DevOps

Azure DevOps services are used during development and deployment of integrations.

Development Process

1. Get IntegrationID according to Naming conventions from the central register of integrations.
2. Identify the repo to be use. It might be an existing repo according to the central register of integrations. If a new repo is needed create the repo and follow naming conventions. TODO: Naming?
3. Secret information e.g. usernames, passwords, subscription keys, certificates should never be commited into a repo. Use Azure Key Vault for storing secrets.
4. Use Managed Identity when possible for securing communication between Azure services. Use role-based access control (RBAC) to grant permissions.
5. Always use feature branches and merge to master after a pull request.
6. Always build and deploy integrations with Pipelines. A Pipeline should deploy to all target environments with approvals and checks setup.

TODO: Decide how pipelibe should be setup?

TODO: decide naming conventions for repos.

Repo

All source code should be commited and pushed into a repo. A repo group one or more integrations related to a domain or process. In addition, integrations related to a business applications relation can be grouped e.g. all integrations between system A and system B.

The main folder structure is described in the next section Delivery Pipeline.

Delivery Pipeline

All deliveries of integration artifacts should be orchestrated with Azure Pipelines.

The Delivery pipeline is divided into three parts. The main part which is triggered when new code is pushed to the repository. The stages of the delivery pipeline are implemented here. It is defined in a file called azure-pipelines.yaml. The commit pipeline implements all jobs that should be run to prepare the artifacts for deployment, like compile, validate and test. It is defined in a file called commit.yaml The release pipeline implements all jobs that deploy artifacts to a specified environment. It is defined in a file called release.yaml.

```
.
+-- .pipeline
|   +-- commit.yaml
|   +-- release.yaml
+-- apis
+-- azure-pipelines.yaml
+-- components
+-- resources
```

Integrio has gathered their best practices into reusable pipeline templates. This standardization minimizes variations and potential errors that can arise from manual configurations or individual interpretations of documentation.

The library and how to use it can be found here: Pipeline Templates

Bicep Modules

Integrio har gathered their best practices into reusable Bicep modules that ensure that every piece of infrastructure is built to the same standards. This standardization minimizes variations and potential errors

that can arise from manual configurations or individual interpretations of documentation.

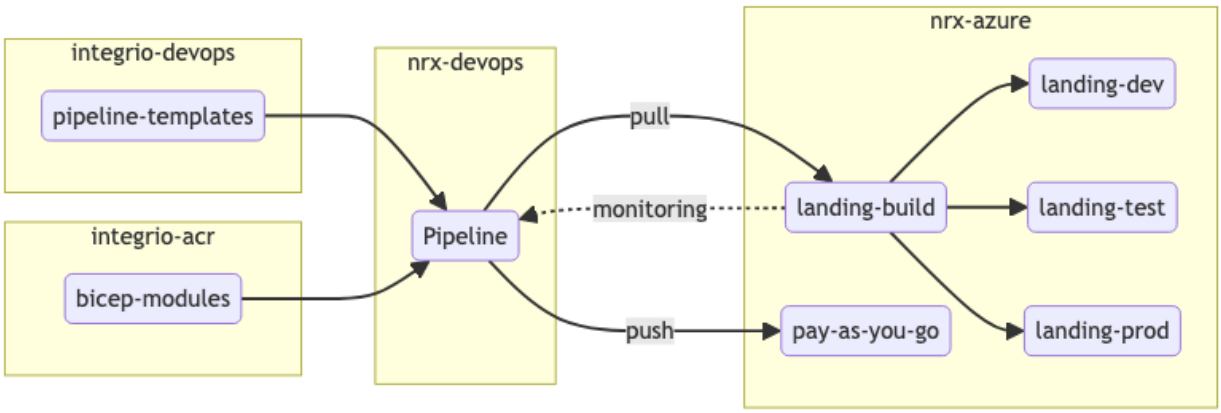
The library and how to use it can be found here: Bicep Modules

Service Connections

The Azure Pipelines that deploys integrations need Service Connections to be able to create and/or updates Azure resources. The existing Service Connections that should be used are listed in the table below.

Name	Target Environment	Description
nrx-integrations-dev-sp	Landing Zone Development	Used to deploy integrations in the DEV environment.
nrx-integrations-test-sp	Landing Zone Test	Used to deploy integrations in the TEST environment.
nrx-integrations-prod-sp	Landing Zone Production	Used to deploy integrations in the PROD environment.
nrx-integrations-payg-sp	Betala per användning	Used to deploy integrations in the old Azure environment (Dev, Staging, Prod).
igr-integration-acr-sp	Integrio Infra	Used to reference Bicep modules published in Integrio ACR.
igr-integration-devops-sp	Integrio DevOps	Used to reference Pipeline templates in Integrio DevOps.

Service Connections Overview



Azure Functions

Azure Functions is a serverless compute service that enables you to run event-triggered code without having to explicitly provision or manage infrastructure. Using Azure Functions, you can run a script or piece of code in response to a variety of events. Azure Functions can be used to process data, integrate systems, trigger alerts, and more.

CAF naming conventions

Azure Functions should follow the CAF naming conventions.

Programming languages

TODO: we need to decide on which programming languages to support in Azure Functions.

Storage for Azure Functions

TODO: if we decide that each function needs its own storage account, we need to follow CAF naming conventions for storage accounts. We also need to figure out pricing related to storage accounts.

Infrastructure as Code

IaC, Infrastructure as Code, gives the ability to always be able to deploy the same code that has been committed in the repository. Two languages that enables this (together with other tooling) are Bicep and YAML.

Bicep

A recommended way to deploy Azure resources and infrastructure is to use Azure Bicep as IaC (Infrastructure as Code). Bicep gives the advantage of modularity and reusability, and will be compiled and translated into ARM templates (Azure Resource Manager) when deployed. ARM code is relatively verbose and difficult to write, Bicep has a much simpler syntax.

There is a lot of documentation on how to use Bicep, for example: - Microsoft Bicep documentation: https://learn.microsoft.com/en-us/azure/azure-resource-manager/bicep/ - Microsoft Fundamentals of Bicep: https://learn.microsoft.com/en-us/training/paths/fundamentals-bicep/

In order to develop Bicep code on a developer machine, it is recommended to install and use VS Code and the VS Code Bicep extension.

As mentioned, Bicep has the advantage of being able to organize into modules. Bicep is also idempotent, and the same bicep file can be deployed multiple times if needed, and you will get the same resources and in the same state every time. It is perfectly possible to write Bicep code all from scratch, and even create your own modules. However, to alleviate some of this work (and at the same time get tried-and-tested bicep code), there is also the possibility to use existing modules. These modules exist in a separate repository and can be referenced from your Bicep code. For more information, see Integrio Bicep Modules

YAML

YAML is used when writing the pipeline that constructs the Devops CI/CD chain. There are also pre-created YAML-templates that exists in another separate repository, which can be referenced from your YAML files. For more information, see Integrio Pipeline Templates

---

Terraform

TBD.

Development security

TODO: describe how we handle security in development (service principals, keyvault, etc.)

This document describes:

- service principals
- keywault

Service principals

Keyvault

CAF Naming conventions

We are using CAF naming conventions. The purpose of having defined naming standards is to make it possible to identify what a resource does and where it belongs just by looking at the name.

CAF naming convention for the Azure Functions

We have a central register of all integrations. The IntegrationID is a unique identifier for each integration. The IntegrationID is used in the name of the function app.

The format of the integration ID is "int" followed by a three-digit number. The number is unique for each integration.

TODO: we must make final decition on naming. Can the integration number can be added as a tag?

Functions need to be named in a CAF compliant way:

Name: func-api-testfunction-int0001-euw

Key-word	Example	Chars	Description
	`func`	3	Indicates the resource is a Function App.
	`api`	3	Denotes the integration landing zone.
	`testfunction`	12	Specifies the name of the function app.
	`int001`	6	The unique IntegrationID from our tracking system.
	`eus`	3	The region abbreviation for East US.
	`01`	2	A two-character hexadecimal to make the name unique. An instance or sequence number for versioning or multiple instances.

CAF naming convention for Azure Storage accounts

Azure Storage account names must be between 3 and 24 characters in length and can contain only lowercase letters and numbers. And it ***must be unique across all of Azure***.

TODO: Storage accounts are hard to name as they must be unique across all of Azure. What is the best practice for naming?

Because of the lenght of maximum 24 characters we have limited the length of the integration ID to 6 characters.

Name: stapitestfnint001eus01

Key-word	Exam-ple	Chars	Description
	`st`	46	2 char that denotes it's a storage account.
	`api`	64	Max 4 char that indicate landing zone. Indicates it's part of the API landing zone.
	`testfn`	22	6 char for the name
	`int001`	24	6 char Integration ID
	`eus`	61	3 char that Specifies the Azure region (East US) for the storage account.
	`01`	60	A two char hex to make the name unique.

The above rule will use the maximum length of 24 characters.

Functions have separate storage accounts so that different teams do not interfere with each other. This is a best practice for isolation and security.

TODO: What is the costs associated with storage accounts? Should we have separate storage accounts for each function app?

Costs

This is an overview of the costs for the infrastructure for Red Cross Norway.

Product	Description	Version	Monthly Cost
Azure Application Gateway	Firewall		*?1
Azure Web Application Firewall (WAF)	Firewall - exploits protection	included	
Azure DDoS Protection	Firewall - DDoS protection	Basic is included	
Azure API Management	APIM prod	API Management, Standard v2	kr7,523
Azure API Management	APIM test	API Management, Standard v2	kr7,523
Azure API Management	APIM dev	API Management, Standard v2	kr7,523 *?2
Landing Zone Prod	Landing zone for production		
Landing Zone Test	Landing zone for testing		
Landing Zone Dev	Landing zone for development		
Landing Zone Build	Landing zone for build server		
Build server	Linux VM that builds and deploys		

Questions:

1) “Azure Application Gateway: Overview: Azure Application Gateway is a platform-managed, scalable, and highly available application delivery controller (ADC) as a service. It provides layer 7 load balancing, centralized SSL offload, and integrated web application firewall (WAF) capabilities.” Azure DDoS Protection Basic is included, Standard is an extra service. It seems that all the stuff we need is in the same product. We already have a firewall in the new CleanAzure subscription. It is named afw-prod-hub-network-euw so we do not need another one!

2) Can we use the API Management “Basic” or developer tier instead of “Standard v2” for the dev landing zone? See costs for API Management here

HOWTO Document

This is how we write the documentation.

Documentation is split up in sevaral Markdown files. Each file is a chapter in the documentation. The fantastic thing about Markdown is that it is easy to write and easy to read. It is also easy to convert to other formats like HTML, PDF, and Word.

Diagrams are text. We use Mermaid to create diagrams. Mermaid is a simple markdown-like script language for generating charts that is visible in the markdown editor. This makes it easy to maintain the documentation and the diagrams in the same file.

Azure DevOps has a built-in wiki that supports markdown and you can read about it here Azure DevOps Wiki support for Mermaid.

howto create diagrams

We use mermaid to make diagrams. The diagrams are defined in markdown files and rendered by the mermaid plugin in the markdown editor. This makes it possible to define diagrams in the same file as the text that describes the diagram. We follow the guide defined by Diagram Guide by kubernetes community.

- Why use mermaid? Because it is easy to use and it is rendered in the markdown editor.
- Why use mermaid? Because documentation and diagrams are in the same file. And they are versioned together (git).
- Why not use draw.io? Because it is not rendered in the markdown editor and you need to use a separate program to maintain the diagram.
- Why not use powerpoint? Because it is not rendered in the markdown editor and you need to use a separate program to maintain the diagram.

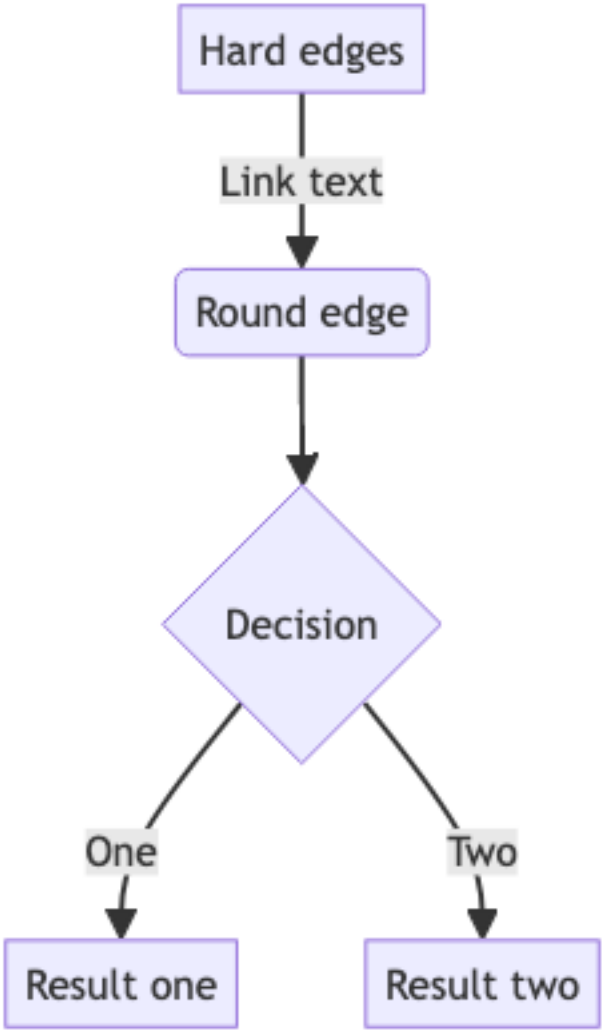
Important In Azure DevOps the notation is “`but in the free world it is` Our doc is in DevOps so you must use the “‘ notation.

KISS - Keep It Simple Stupid

Keep the diagrams simple. Use the same style for all diagrams. Use the mermaid live editor to create the diagrams. It is easier to create the diagrams in the live editor and then copy the code to the markdown file. You find the mermaid live editor at mermaid live editor. You can find the full documentation here. If you use vscode as your editor you can view the diagrams in the preview mode.

If you want to dep dive and see how it compares to other tools read the mermaid.js: A Complete Guide

Example 1



The diagram above is created from this text:

```
graph TD
  A[Hard edges] -->|Link text| B(Round edge)
  B --> C{Decision}
  C -->|One| D[Result one]
  C -->|Two| E[Result two]
```

Example 2

The text that creates the diagram above is:

```
graph LR;
 client([client])-. Ingress-managed <br> load balancer .->ingress[Ingress];
 ingress-->|routing rule|service[Service];
 subgraph cluster
 ingress;
 service-->pod1[Pod];
 service-->pod2[Pod];
 end
 classDef plain fill:#ddd,stroke:#fff,stroke-width:4px,color:#000;
 classDef k8s fill:#326ce5,stroke:#fff,stroke-width:4px,color:#fff;
 classDef cluster fill:#fff,stroke:#bbb,stroke-width:2px,color:#326ce5;
 class ingress,service,pod1,pod2 k8s;
 class client plain;
 class cluster cluster;
```

howto edit Wiki in Visual Studio Code

It is possible to edit Azure DevOps wiki pages using Visual Studio Code (VSCode). This can make editing more efficient, especially if you prefer VSCode's editing environment, extensions, and tools. Here's how you can set it up and start editing your wiki pages in VSCode:

- Step 1: Clone the Wiki Repository

Azure DevOps wikis are backed by a Git repository, making them easy to clone and edit locally.

Navigate to your Wiki in Azure DevOps: Go to the Wiki section of your Azure DevOps project. Find the Clone URL: In the Wiki section, there should be an option to clone the wiki. Copy the URL provided. Clone using VSCode: Open VSCode. Open the Command Palette (Ctrl+Shift+P or Cmd+Shift+P on macOS). Type Git: Clone and paste the URL you copied. Select the local directory where you want to clone the wiki repository.

- Step 2: Edit the Wiki Pages

Once the wiki is cloned, you can navigate through the folder structure in VSCode. Wiki pages are usually markdown files (.md), so you can open and edit them as you would with any markdown document in VSCode.

- Step 3: Commit and Push Changes

After editing the wiki pages, you need to commit and push your changes back to the Azure DevOps repository for them to be reflected in the online wiki.

Commit your changes: Open the Source Control view in VSCode (the branch icon on the left panel). Stage your changes by clicking on the + icon next to the edited files. Enter a commit message and press Ctrl+Enter (or Cmd+Enter on macOS) to commit the changes locally. Push your changes: Click on the three dots in the Source Control view and select Push to push your changes to Azure DevOps.

- Step 4: Sync Changes

If multiple people are editing the wiki, ensure you pull the latest changes before starting your editing session to minimize merge conflicts. You can do this from the Source Control view in VSCode by pulling from the repository.

- Additional Tips

Use Extensions: Enhance your markdown editing experience in VSCode by installing extensions like "Markdown All in One" or "MarkdownLint" for better linting and productivity features. Preview Changes: VSCode allows you to preview Markdown files with a live preview window (Ctrl+Shift+V or Cmd+Shift+V on macOS), which can be very useful when editing wiki pages. By using VSCode, you can leverage a more powerful editor for managing your Azure DevOps wiki pages, along with the comfort and familiarity of your local development environment.

howto output one PDF that contains all the documentation

The documentation is split up in several markdown files. This is great for maintaining the documentation. But sometimes you need to have the documentation in one file. To create one PDF that contains all the documentation you need to merge the markdown files into one file and then convert it to PDF.

TODO: Create a service that run every time a commit is made to the repository. The service will merge the markdown files and convert it to PDF. The PDF will be stored in the repository and can be downloaded from the repository.

Notes from testing:

1. Install Pandoc

- On Windows, you can use Chocolatey: `choco install pandoc` ????=> not tested
- On Mac, you can use Homebrew: `brew install pandoc`

2. Install the mermaid filter for Pandoc

- `npm install -g mermaid-filter`

3. install a later version of pdf creator named lulatex installed BasicTeX.pkg from https://tug.org/mactex/morepackages.html sudo tlmgr update –self sudo tlmgr install datetime sudo tlmgr install collection-latexrecommended sudo tlmgr install collection-latexextra sudo tlmgr install selnolig

4. INstall the latex for creating PDF documents

- on Mac `brew install --cask mactex`
- on Windows `choco install miktex` ????=> not tested

4. Install pandoc-crossref in order to reference figures (did not solve the display of figures problem)

- on Mac `brew install pandoc-crossref`
- on Windows `choco install pandoc-crossref` ????=> not tested

Generated list of TODO:s

Warning: This document contains a list of TODO:s extracted from the markdown files. Do not edit this file directly. Generated at Fri Apr 26 21:32:30 CEST 2024

Section	TODO Item
Firewall documentation	TODO: Describe the configuration of the firewall.
Firewall SSL termination	TODO: Jah - is the certificate installed on the FW in the new subscription or the old?
Firewall Logging	TODO: Add information about logging and who is responsible for monitoring the logs.
Landing Zone Development	TODO: The dev landing zone can only be accessed from the internal network. How is this done?
Definition of external systems	TODO: we need to make everyone aware of the consequences of this.
Communication rules exceptions	TODO: If there should ever be exceptions to the comminication rules. We ned to document how we make exceptions to the rules.
How to set up the build server for a repository	TODO: Describe how to set up the build server for a repository.
Development Process	TODO: Decide how pipelibe should be setup?
(No heading)	TODO: decide naming conventions for repos.
Programming languages	TODO: we need to decide on which programming languages to support in Azure Functions.
Storage for Azure Functions	TODO: if we decide that each function needs its own storage account, we need to follow CAF naming conventions for storage accounts. We also need to figure out pricing related to storage accounts.
Development security	TODO: describe how we handle security in development (service principals, keyvault, etc.)
CAF naming convention for the Azure Functions	TODO: we must make final decition on naming. Can the integration number can be added as a tag?
CAF naming convention for Azure Storage accounts	TODO: Storage accounts are hard to name as they must be unique across all of Azure. What is the best practice for naming?
(No heading)	TODO: What is the costs associated with storage accounts? Should we have separate storage accounts for each function app?
howto output one PDF that contains all the documentation	TODO: Create a service that run every time a commit is made to the repository. The service will merge the markdown files and convert it to PDF. The PDF will be stored in the repository and can be downloaded from the repository.