TITLE

SpaceCloud Framework

PROJECT

InCubed Phase 1

Action	Name	Function	Date
Author:	Aris Synodinos	System Engineer	10/28/2020
Reviewed by:			
Checked by:			
Authorized by:			

Copying of this document, and giving it to others and the use or communication of the contents, thereof, is forbidden without express authority. Offenders are liable to the payment of damages. All rights are reserved in the event of the grant of a patent or the registration of utility model or design.

Unibap AB (publ.)
Kungsängsgatan 12
SE-753 22 Uppsala
Sweden
Tel: +46-18-320 330
info@unibap.com
unibap.com

Kopiering av detta dokument, delgivande till andra samt använda eller kommunicera innehållet, är förbjudet utan specifikt tillstånd. Brott mot detta kan lede till skadeståndskrav. Alla rättigheter är reserverade i händelse av patent eller registrering av design.

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

Distribution List

Name	No. of Copies	Company / Organization

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

Document Change Record

Issue	Date	Page	Comments	Author
1	10/28/2020	-	Initial issue	ArSy

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

Table of Contents

1 INTRODUCTION 7
1.1 Intended Audience 7
2 DOCUMENTS 8
2.1 Applicable Documents 8
2.2 Reference Documents 8
2.3 Acronyms & Abbreviation List 9
2.4 Definitions 9
3 ARCHITECTURE 10
3.1 SCFW Roles 10
3.1.1 Administrator 10
3.1.2 Developer 10
3.1.3 User 10
3.2 Capabilities 11
3.2.1 Computation 11
3.2.2 Concurrency 11
3.2.3 Storage 12
3.2.4 Sensors 12
3.2.5 Communication 12
4 APPLICATION DEVELOPMENT 13
4.1 Accessing the SDK 13
4.2 Downloading the SDK 13
4.3 Creating a basic app – Hello world! 13
4.3.1 Dockerfile 13
4.3.2 Build the app 14
4.4 SpaceCloud Framework API 14
4.5 Accessing the sensors 17
4.5.1 Making gRPC calls 17
4.5.2 Build the app 17
4.5.3 Deserializing the image 18
4.5.4 Location 19

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

4.6 Getting persistent storage ... 20

4.7 Setting up communication ... 21

4.8 Running multiple applications ... 22

4.9 Releasing new applications ... 23

4.9.1 Tagging the application ... 24

4.9.2 Uploading the application ... 24

4.10 Advanced Usage ... 24

5 APPLICATION TESTING ... 25

5.1 Execution parameters ... 25

5.2 Execution results ... 26

5.3 Conformance testing ... 27

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

List of Figures

Figure 1 SpaceCloud Framework 7
Figure 2 SCFW simplified model 10
Figure 3 SCFW Node hardware 11
Figure 4 Generic Sensor Interface abstraction layer 12
Figure 5 The SCFW Conformance Testing suite 27

List of Tables

Table 2-1 Applicable Documents 8
Table 2-2 Reference Documents 8
Table 2-3 Acronyms & Abbreviation List 9
Table 2-4 Definition List 9

List of Code Blocks

Code 1 Login the SCFW registry 13
Code 2 Hello world dockerfile 14
Code 3 Building an application 14
Code 4 Build output 14
Code 5 Locally running an application 14
Code 6 SCFW Proto API 15
Code 7 RunApplication message 15
Code 8 SetStorageSynchronization message 15
Code 9 GetImage message 17
Code 10 GetLocation message 17
Code 11 Requesting an Image 17
Code 12 Dockerfile for sensors demo 18
Code 13 Image converter python 19
Code 14 Image converter dockerfile 19
Code 15 Get location python 20
Code 16 Persistent storage 21
Code 17 ARMS communication python 22
Code 18 Run concurrent application python 23
Code 19 Docker tag 24
Code 20 Docker push 24
Code 21 Multistage building 24
Code 22 Execution parameters 26

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

1 Introduction

The SpaceCloud Framework revolutionizes satellite software development, converting purpose built custom space hardware into flexible and reusable compute nodes.

Most sensor data from satellites gets downlinked to earth, where it gets distributed to either HPC clusters for processing or gets forwarded to one of the major cloud processing providers. There are some disadvantages in this approach, but the processing power in most satellites and the difficulty of development and deployment for space missions has always been a limiting factor for advancement.

The Unibap SpaceCloud™ line of products includes both the hardware and the software platform to simplify space missions, providing an all-inclusive solution for data processing in orbit and management software on ground.

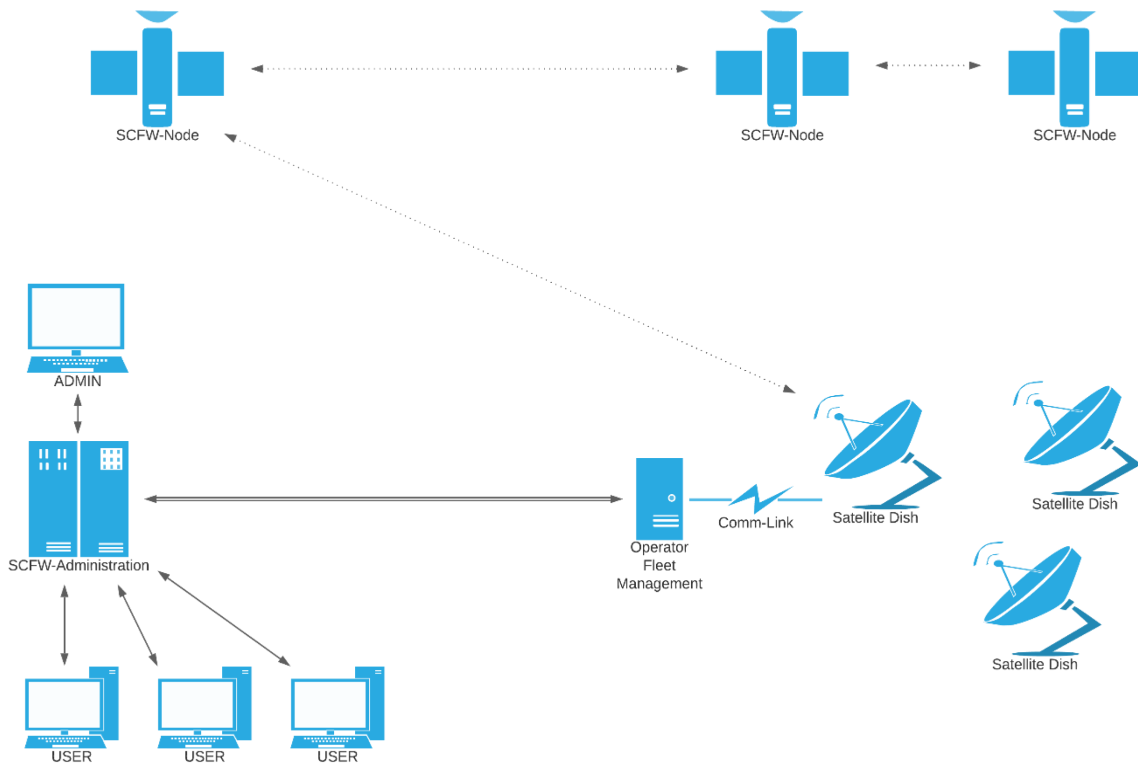

Figure 1 SpaceCloud Framework

1.1 Intended Audience

This manual is intended for users who want an introduction to the SCFW, either to understand the software concepts introduced in the platform or for getting hands-on with developing SCFW applications. The SCFW is currently in alpha testing phase, with all tests performed in a controlled environment on Unibap AB premises. The following document describes the process for developing and testing applications during this phase of the framework and does not represent the final form of the SCFW. Both the API as well as all the processes themselves can be changed depending on feedback from users as well as internal development.

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

2 Documents

2.1 Applicable Documents

This document shall be read in conjunction with documents listed hereafter, which form part of this document to the extent specified herein. In case of a conflict between any provisions of this document and the provisions of the documents listed hereafter, the content of the contractually higher document shall be considered as superseding.

AD	Doc. No.	Issue	Rev.	Title
[AD1]				
[AD2]				
[AD3]				
[AD4]				
[AD5]				

Table 2-1 Applicable Documents

2.2 Reference Documents

The following documents contain additional information that is relevant to the scope of this document.

RD	Doc. No.	Issue	Rev.	Title
[RD1]				
[RD2]				
[RD3]				
[RD4]				

Table 2-2 Reference Documents

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

2.3 Acronyms & Abbreviation List

Abbreviations are listed in Table 2-3 of this document

Abbreviations	
UAB	Unibap AB (publ.)
SCFW	SpaceCloud Framework
SCHW	SpaceCloud Hardware
SDK	Software Development Kit
API	Application Programming Interface
RPC	Remote Procedure Call
OCI	Open Container Initiative

Table 2-3 Acronyms & Abbreviation List

2.4 Definitions

Definitions are listed in Table 2-4 of this document.

Definitions	
SCFW Context	The isolated environment where all applications of a user get executed

Table 2-4 Definition List

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

3 Architecture

The software stack of the SCFW abstracts the hardware and to some extent the operating system from satellite applications. A deployed SCFW system is comprised of two different types of systems, the administration system and the computational nodes.

There are different roles for interacting with the SCFW, each role having different permissions and capabilities. All interaction uses the SCFW Administration as the entry point, the SCFW Nodes are self-contained systems that do not interact directly with any of the specified roles, but only indirectly through the specified API.

Figure 2 SCFW simplified model

3.1 SCFW Roles

There are three predefined roles on a typical SCFW Administration server. Each account on the platform can be a member of any number or roles, enabling different permissions and functionality for each of those.

3.1.1 Administrator

The administrator role is responsible of administering the SCFW Nodes that are managed by the installation. He specifies what operations each node will perform, including approving execution of applications and upgrading the SCFW software of each node. He is also responsible of performing the initial installation of the SCFW Node software on the actual SCHW.

3.1.2 Developer

The developer role can upload new applications to the SCFW AppStore. As an independent developer, one would develop generic applications that provide some generic capability – easily configurable for end users (e.g. a cloud detecting application). Those applications would be customer agnostic and accessible publicly – either freely or under some licensing contract.

As a mission specific developer, one would develop private applications that cannot be used by other users. He can then select the specific SCFW accounts that the application will be accessible and deployable from.

3.1.3 User

The user role describes the end-user of the platform. As a user, one can request applications to execute on SCFW Nodes. The user defines which applications he wants to execute, when to execute them and on what resources. The user defines all the configuration of application execution, including storage, access to CPU, GPU or other hardware accelerators. Access to sensors as well as

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

communications interfaces are also defined by the user. The user does not decide what gets deployed or not on an actual SCFW Node, that is the responsibility of the administrator, however the user is the role that makes the application execution request.

3.2 Capabilities

- CPU, GPU, VPU access for computation
- Concurrency of multiple applications in a sandboxed environment
- Persistent and Temporary isolated storage space for each account
- Unified sensors API
- Unified Communication API

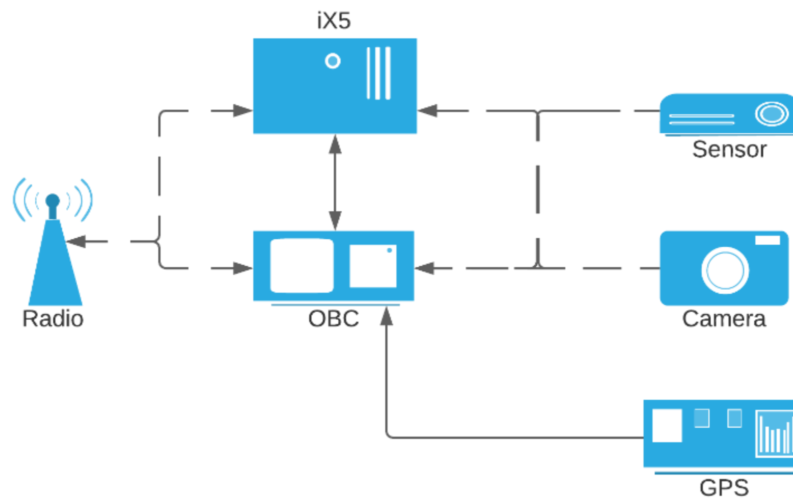

Figure 3 SCFW Node hardware

3.2.1 Computation

The SCFW is designed with the limitations of space operation in mind. Given that, applications deployed on nodes can utilize the SCHW computational capabilities to the maximum, with an insignificant amount of overhead compared to bare-metal application development.

Access to CPU, GPU and even VPU can be enabled for developed SCFW Apps available to the SCFW Container Registry. Depending on the actual hardware capabilities of the node, a user can utilize the devices optimally to perform efficiently computations directly on the satellite and reduce the latency from data gathering to actionable decisions.

3.2.2 Concurrency

The SCFW makes sharing node resources simple. When a user specifies the SCFW Context, he chooses if during execution, other SCFW Contexts can be launched concurrently. This allows multiple users to execute software concurrently on the same Node in complete isolation.

To complement that, concurrency is also supported within the context as well as. A user can specify multiple applications to be spawned automatically or manually within his SCFW Context, allowing him to communicate over IP through a virtual networking device with hostname

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

resolution. Complex pipelines can therefore be accommodated with off-the-shelf applications supplied by experienced developers – cloud detection, fire detection or other applications that can be deployed to the next SCFW-enabled satellite.

3.2.3 Storage

Both persistent and temporary storage are provided in SCFW applications in the form of simple file storage. This allows developers and users to utilize their existing codebase that assumes normal filesystem access to their SpaceCloud applications.

3.2.4 Sensors

Satellite sensors vary significantly depending on mission objectives and hardware architecture. The SCFW aims to reduce the complexity of developing highly specialized satellite software that only executes on one specific platform by providing a generic abstract API for common types of sensors.

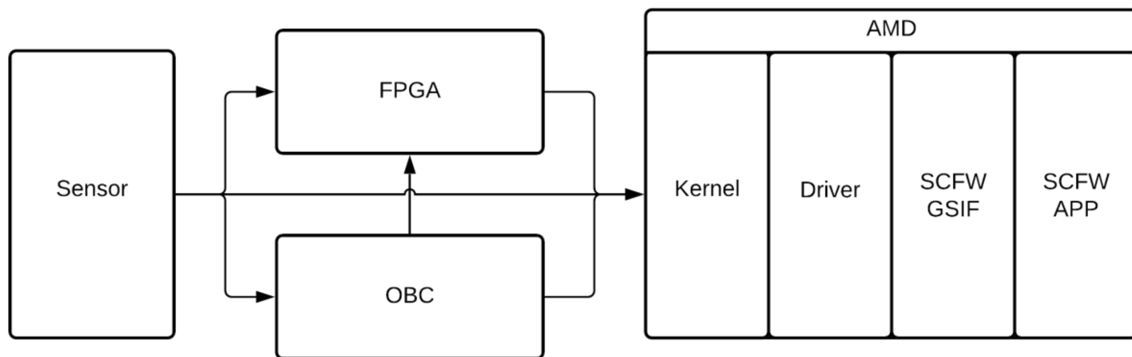

Figure 4 Generic Sensor Interface abstraction layer

3.2.5 Communication

In order to simplify radio communication for app developers and users, the SCFW uses the persistent file system storage as the data layer for communication. App developers and users no longer need to be developing or maintaining complex communication frameworks and serialization/deserialization algorithms. With the SCFW, you save your files into the persistent storage volume and notify that you want that volume to be synchronized down or up. On the next synchronization event, you will have your files synced to your favorite cloud provider, easy to access and process without having to learn new libraries.

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

4 Application Development

The SCFW utilizes container images and containers for encapsulating SCFW applications. In order to create applications, it is useful to have some familiarity with one of the popular OCI tools, either docker or podman. Due to the larger userbase, the examples included in this manual will use docker, but replacing docker with podman should be simple and most likely a drop-in replacement.

The installation of those tools is not covered in this manual and can be found in the documentation of the tools themselves for various operating systems. It is recommended to use a GNU/Linux computer as a development machine. Although docker is supported in Windows 10, some functionality may be experimental or even completely unsupported.

4.1 Accessing the SDK

In order to access the SDK, you must have an account to access the SCFW container registry. To create an account, you must send an email to spacecloud@unibap.com with the subject “SCFW Account” and in the body the username, full name and company name.

Once the account has been created, you can login to https://spacecloud.unibap.com/ in order to be able to download the SDK.

4.2 Downloading the SDK

Once you have obtained the username and password for the SCFW container registry, you can use that to permanently login your development machine to docker. To do that, in a terminal issue the following command

```
docker login spacecloud.unibap.com
```

Code 1 Login the SCFW registry

It will then ask you the username and password and once these are entered, you will be able to pull and push container images to the registry.

4.3 Creating a basic app – Hello world!

There are many ways to create a new container image, and not all will be covered in this manual. The following instructions will guide the user step-by-step into creating a simple SCFW Application, other methodologies are also compatible if they create valid container images. For more complex use cases, one can read on section 4.10.

4.3.1 Dockerfile

To create a container image, you should create a description file that your preferred image builder will parse and attempt to generate your application. Docker and Buildah, the most popular tools for building images, support dockerfiles as input.

The first line of a dockerfile is typically the FROM statement. This defines the base image that is used for generating your application. It is recommended to use the SCFW Runtime image for your application, as it will limit the bandwidth required to upload your application to SCFW Nodes.

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

The following two lines are simple and self-explanatory, SHELL defines the shell that will be used for the subsequent instructions during the build of the container image and CMD defines which command will run when the container images will be executed. In this example, command echo will run with argument the string “Hello Spacecloud!”.

```
FROM spacecloud.unibap.com/unibap/framework-runtime
SHELL ["/bin/bash", "-c"]
CMD ["echo", "Hello Spacecloud!"]
```

Code 2 Hello world dockerfile

4.3.2 Build the app

To create a container image from this dockerfile, you issue the following command in a terminal

```
docker build .
```

Code 3 Building an application

Which generates the following output

```
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM spacecloud.unibap.com/unibap/framework-runtime
 ---> 4791e059cb7e
Step 2/3 : SHELL ["/bin/bash", "-c"]
 ---> Running in 0fdb5296b621
Removing intermediate container 0fdb5296b621
 ---> 767145655802
Step 3/3 : CMD ["echo", "Hello Spacecloud!"]
 ---> Running in 6f99b5f1b68a
Removing intermediate container 6f99b5f1b68a
 ---> 76f9b9e4a8ba
Successfully built 76f9b9e4a8ba
```

Code 4 Build output

To execute the container image generated, you can use the generated hash id as following

```
docker run 76f9b9e4a8ba
Hello Spacecloud!
```

Code 5 Locally running an application

You can pull this container image at spacecloud.unibap.com/unibap/hello-world:latest

4.4 SpaceCloud Framework API

Although direct access to computational resources is feasible for SCFW applications, all other common resources on SCFW Nodes are accessed indirectly with RPC. This allows access to the sensor data and all functionality provided by the SCFW through multiple different programming languages.

The gRPC framework is chosen, due to the simplicity, accessibility and popularity. The API of the SCFW can therefore be defined using protobuf description files that can be parsed and create language specific implementation files. This manual does not dive into the details of the gRPC framework, as the documentation provided by the developers is very extensive and easily accessible.

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

There are two gRPC servers handling incoming requests from executing applications. The ARMS server handles Resource Management on port 50001 while the sensors server handles all requests for sensor data on port 50002. The IP that must be used to ensure reliable communication to the SCFW is 172.17.0.1 on an insecure channel.

The protobuf file description for the supported RPC methods are described in the following code block.

```
syntax = "proto3";

import public "google/protobuf/timestamp.proto";

package spacecloud;

service ARMS {
  rpc RunApplication(RunApplicationRequest) returns
(RunApplicationResponse) {}
  rpc SetStorageSynchronization(SetStorageSynchronizationRequest) returns
(SetStorageSynchronizationResponse) {}
}

service Sensors {
  rpc GetImage(GetImageRequest) returns (GetImageResponse) {}
  rpc GetLocation(GetLocationRequest) returns (GetLocationResponse) {}
}
```

Code 6 SCFW Proto API

While the messages for those services are defined in the following codeblocks

```
message RunApplicationRequest {
  string hostname = 1;
}

message RunApplicationResponse {}
```

Code 7 RunApplication message

```
message SetStorageSynchronizationRequest {
  enum ModeType {
    SYNCHRONIZE_UPLINK_ERASE      = 0;
    SYNCHRONIZE_UPLINK_MAINTAIN   = 1;
    SYNCHRONIZE_DOWNLINK_ERASE    = 2;
    SYNCHRONIZE_DOWNLINK_MAINTAIN = 3;
  }
  string name = 1;
  int32 priority = 2;
  ModeType mode = 3;
}

message SetStorageSynchronizationResponse {}
```

Code 8 SetStorageSynchronization message

```
message GetImageRequest {}
```

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

```
message GetImageResponse {
  google.protobuf.Timestamp stamp = 1;
  uint32 height = 2;
  uint32 width = 3;
  enum EncodingType {
    ENCODING_TYPE_UNKNOWN = 0;
    ENCODING_TYPE_RGB8 = 1;
    ENCODING_TYPE_RGBA8 = 2;
    ENCODING_TYPE_RGB16 = 3;
    ENCODING_TYPE_RGBA16 = 4;
    ENCODING_TYPE_BGR8 = 5;
    ENCODING_TYPE_BGRA8 = 6;
    ENCODING_TYPE_BGR16 = 7;
    ENCODING_TYPE_BGRA16 = 8;
    ENCODING_TYPE_MONO8 = 9;
    ENCODING_TYPE_MONO16 = 10;
    ENCODING_TYPE_8UC1 = 11;
    ENCODING_TYPE_8UC2 = 12;
    ENCODING_TYPE_8UC3 = 13;
    ENCODING_TYPE_8UC4 = 14;
    ENCODING_TYPE_8SC1 = 15;
    ENCODING_TYPE_8SC2 = 16;
    ENCODING_TYPE_8SC3 = 17;
    ENCODING_TYPE_8SC4 = 18;
    ENCODING_TYPE_16UC1 = 19;
    ENCODING_TYPE_16UC2 = 20;
    ENCODING_TYPE_16UC3 = 21;
    ENCODING_TYPE_16UC4 = 22;
    ENCODING_TYPE_16SC1 = 23;
    ENCODING_TYPE_16SC2 = 24;
    ENCODING_TYPE_16SC3 = 25;
    ENCODING_TYPE_16SC4 = 26;
    ENCODING_TYPE_32SC1 = 27;
    ENCODING_TYPE_32SC2 = 28;
    ENCODING_TYPE_32SC3 = 29;
    ENCODING_TYPE_32SC4 = 30;
    ENCODING_TYPE_32FC1 = 31;
    ENCODING_TYPE_32FC2 = 32;
    ENCODING_TYPE_32FC3 = 33;
    ENCODING_TYPE_32FC4 = 34;
    ENCODING_TYPE_64FC1 = 35;
    ENCODING_TYPE_64FC2 = 36;
    ENCODING_TYPE_64FC3 = 37;
    ENCODING_TYPE_64FC4 = 38;
    ENCODING_TYPE_BAYER_RGGB8=39;
    ENCODING_TYPE_BAYER_BGGR8=40;
    ENCODING_TYPE_BAYER_GBRG8=41;
    ENCODING_TYPE_BAYER_GRBG8=42;
    ENCODING_TYPE_BAYER_RGGB16=43;
    ENCODING_TYPE_BAYER_BGGR16=44;
    ENCODING_TYPE_BAYER_GBRG16=45;
    ENCODING_TYPE_BAYER_GRBG16=46;
    ENCODING_TYPE_YUV422=47;
  }
  EncodingType encoding = 4;
  bool is_bigendian = 5;
  uint32 step = 6;
  bytes data = 7;
}
```

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

Code 9 GetImage message

```
message GetLocationRequest {
  google.protobuf.Timestamp stamp = 1;
}

message GetLocationResponse {
  double latitude = 1;
  double longitude = 2;
  double elevation = 3;
  double orbital_speed = 4;
}
```

Code 10 GetLocation message

4.5 Accessing the sensors

Following the definitions of the SCFW API, it is now possible to create an application that utilizes that to communicate with the sensors to request an image from the satellite. For simplicity, python3 will be used for the development of the applications of this manual. However, as expected, all officially supported programming languages can be used for communicating with the SCFW Node.

4.5.1 Making gRPC calls

Communicating with either the Sensors or ARMS requires two objects in python, a communication channel (gsif_channel in the python code sample) and the client stub (sensors_stub). Through the client stub, a developer can call an RPC function that is stated in the Services as defined by the protobuf file.

```
#!/usr/bin/env python3

import grpc
from proto.unibap_spacecloud_api_pb2 import GetImageRequest
from proto.unibap_spacecloud_api_pb2_grpc import SensorsStub


if __name__ == "__main__":
    print("Started Application")
    gsif_channel = grpc.insecure_channel('172.17.0.1:50002')
    sensors_stub = SensorsStub(gsif_channel)
    try:
        res = sensors_stub.GetImage(GetImageRequest())
    except grpc.RpcError as e:
        print(e)
    print("Application completed")
```

Code 11 Requesting an Image

4.5.2 Build the app

Once the software has been developed, the next step is to create the dockerfile and then build the image in a similar manner as the hello world application. The differences in this example compared to the previous one, is that we must now describe the process that makes the python code that we wrote into an actually executable script, including installing dependencies as well as generating the python classes that gRPC and proto require.

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

A developer can choose whether he wants to perform this in the dockerfile or on his host computer. There are advantages and disadvantages in either solutions, so it is left to the developer to decide which method provides the best results depending on the problem at hand.

In this example, we shall perform all steps in the dockerfile for simplicity.

Some new dockerfile instructions are introduced in the following example. WORKDIR defines the target folder for the docker image from where all commands will be run from and all COPY instructions will copy local artifacts to. It is assumed that the protobuf definitions of the SCFW API are deployed on the proto subfolder of the sensors-demo project. For more detailed documentation of those and all dockerfile commands, one can visit the docker website.

```
FROM spacecloud.unibap.com/unibap/framework-runtime
SHELL ["/bin/bash", "-c"]
RUN apt-get update -qq \
    && apt-get install -qq -y \
    python3 python-virtualenv
WORKDIR /app
COPY proto/*.proto proto/
COPY app.py .
RUN virtualenv venv -p python3 \
    && source venv/bin/activate \
    && python3 -m pip install grpcio grpcio-tools  \
    && python3 -m grpc_tools.protoc --python_out=./ --grpc_python_out=./ -
I=./ proto/unibap_spacecloud_api.proto
CMD ["venv/bin/python", "app.py"]
```

Code 12 Dockerfile for sensors demo

You can pull this container image at spacecloud.unibap.com/unibap/sensors-demo:simple

4.5.3 Deserializing the image

As the image received from the RPC to the sensors_stub is serialized, it must be parsed and converted to a datatype compatible to popular image processing libraries. The protobuf definition of the image includes all required information that one can parse to reconstruct the image himself to his preference. To simplify the process however and ease development for new applications, an image converter is provided that parses the image message and converts it into a numpy array that can be used by OpenCV and potentially more python image processing libraries.

Modifying the sensors-demo python code to use the image_converter, the resulting code is modified only slightly

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

```python
#!/usr/bin/env python3

import cv2
import grpc
from proto.image_converter import NumpyConverter
from proto.unibap_spacecloud_api_pb2 import GetImageRequest
from proto.unibap_spacecloud_api_pb2_grpc import SensorsStub


if __name__ == "__main__":
    print("Started Application")
    gsif_channel = grpc.insecure_channel('172.17.0.1:50002')
    sensors_stub = SensorsStub(gsif_channel)
    try:
        res = sensors_stub.GetImage(GetImageRequest())
        img = NumpyConverter.from_protobuf(res)
    except grpc.RpcError as e:
        print(e)
    print("Application completed")
```

Code 13 Image converter python

The dockerfile is again modified to include the OpenCV dependency as well as the NumpyConverter class.

```dockerfile
FROM spacecloud.unibap.com/unibap/framework-runtime
SHELL ["/bin/bash", "-c"]
RUN apt-get update -qq \
    && apt-get install -qq -y \
    python3 python-virtualenv
WORKDIR /app
COPY proto/*.proto proto/
COPY proto/*.py proto/
COPY app.py .
RUN virtualenv venv -p python3 \
    && source venv/bin/activate \
    && python3 -m pip install grpcio grpcio-tools  opencv-python \
    && python3 -m grpc_tools.protoc --python_out=./ --grpc_python_out=./ -
I=./ proto/unibap_spacecloud_api.proto
CMD ["venv/bin/python", "app.py"]
```

Code 14 Image converter dockerfile

You can pull this container image at spacecloud.unibap.com/unibap/sensors-demo:opencv

4.5.4 Location

The sensors_stub provides access to the location of the SCFW Node at any given time through a similar gRPC. Given that the timestamp of the capture is defined in the image message, one can use that timestamp to query the precise location of the satellite when that image was taken.

The location message does not require deserialization to be parsed.

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

```
#!/usr/bin/env python3

import os
import cv2
import grpc
from proto.image_converter import NumpyConverter
from proto.unibap_spacecloud_api_pb2 import GetImageRequest,
GetLocationRequest
from proto.unibap_spacecloud_api_pb2_grpc import SensorsStub


if __name__ == "__main__":
    print("Started Application")
    gsif_channel = grpc.insecure_channel('172.17.0.1:50002')
    sensors_stub = SensorsStub(gsif_channel)
    try:
        res = sensors_stub.GetImage(GetImageRequest())
        img = NumpyConverter.from_protobuf(res)
        req = GetLocationRequest(stamp=res.stamp)
        loc = sensors_stub.GetLocation(req)
        print(loc)
    except grpc.RpcError as e:
        print(e)
    print("Application completed")
```

Code 15 Get location python

You can pull this container image at spacecloud.unibap.com/unibap/sensors-demo:location

4.6 Getting persistent storage

Access to persistent storage must be setup by the configuration files prior to the execution of the application. Persistent storage can be attached to a container as a mount point, allowing file system access from inside the container.

It is recommended, in order to develop reusable container images, to use environment variables for specifying the mount points of persistent storage when needed. This allows a developed application to be used in different scenarios by multiple users without needing to change the source code of the application.

Modifying the demo python code, the new code is

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

```
#!/usr/bin/env python3

import os
import cv2
import grpc
from proto.image_converter import NumpyConverter
from proto.unibap_spacecloud_api_pb2 import GetImageRequest
from proto.unibap_spacecloud_api_pb2_grpc import SensorsStub


if __name__ == "__main__":
    print("Started Application")
    gsif_channel = grpc.insecure_channel('172.17.0.1:50002')
    sensors_stub = SensorsStub(gsif_channel)
    try:
        res = sensors_stub.GetImage(GetImageRequest())
        img = NumpyConverter.from_protobuf(res)
        storage_path = os.getenv("STORAGE")
        cv2.imwrite(storage_path + "/image.png", img)
    except grpc.RpcError as e:
        print(e)
    print("Application completed")
```

Code 16 Persistent storage

The dockerfile that produces the image is not affected by this change.

4.7 Setting up communication

Communication in the SCFW is tightly connected to the persistent storage. The storage volumes can be tagged to be either UPLINK or DOWNLINK, meaning that they are synced from or to the SCFW Administration. Once the synchronization event has occurred, the data stored in those volumes can be erased or kept, defined by ERASE or MAINTAIN.

Those properties are defined using one of ARMS provided RPC services. ARMS RPC is provided on port 50001. To set this up, similarly to the sensors API, you must create an arms_stub and an arms_channel.

The SetStorageSynchronizationRequest message is comprised of three fields, name, priority and mode. Name is a string that can be set as either the name of the volume or the name of the mount point. In this example it is set as the name of the mounting point, e.g. “/mnt”. Priority can be any 32bit signed integer, setting this number high means that it is very critical for you to synchronize these data on the next synchronization event. Mode defines if the storage is UPLINK, DOWNLINK, ERASE or MAINTAIN.

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

```
#!/usr/bin/env python3

import os
import cv2
import grpc
from proto.image_converter import NumpyConverter
from proto.unibap_spacecloud_api_pb2 import GetImageRequest,
GetLocationRequest, \
    SetStorageSynchronizationRequest
from proto.unibap_spacecloud_api_pb2_grpc import ARMSStub, SensorsStub


if __name__ == "__main__":
    print("Started Application")
    arms_channel = grpc.insecure_channel('172.17.0.1:50001')
    gsif_channel = grpc.insecure_channel('172.17.0.1:50002')
    arms_stub = ARMSStub(arms_channel)
    sensors_stub = SensorsStub(gsif_channel)
    storage_path = os.getenv("STORAGE")
    try:
        res = sensors_stub.GetImage(GetImageRequest())
        img = NumpyConverter.from_protobuf(res)
        cv2.imwrite(storage_path + "/image.png", img)
        req = GetLocationRequest(stamp=res.stamp)
        loc = sensors_stub.GetLocation(req)
        print(loc)
    except grpc.RpcError as e:
        print(e)
    try:
        req = SetStorageSynchronizationRequest(name=storage_path,
priority=10,

mode=SetStorageSynchronizationRequest.SYNCHRONIZE_DOWNLINK_MAINTAIN)
        arms_stub.SetStorageSynchronization(req)
    except grpc.RpcError as e:
        print(e)
    print("Application completed")
```

Code 17 ARMS communication python

You can pull this container image at spacecloud.unibap.com/unibap/arms-demo:comms

4.8 Running multiple applications

As described in section 3.2.2, it is possible to have concurrent applications running in parallel or to be launched optionally depending on conditions. ARMS provides a RPC to facilitate this on the same channel as communication.

This capability can be very useful to minimize deployment size and resource usage as well as modularity in deployment. As a useful scenario, a deployment can be comprised of the following applications.

- The main application - MAIN
- A cloud detecting application that uses OpenCL - CLOUD
- A car detecting application that uses the VPU - CAR

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

At the specified event, MAIN is launched and queries the sensors to receive the image. Once the image is received, MAIN requests to ARMS to launch CLOUD. CLOUD is forwarded the image from MAIN, processes the image and decides if the image is occluded by clouds or not. If the image is not occluded, MAIN asks ARMS to launch CAR in order to process it and save useful data about it. If it is occluded, both CLOUD and ARMS are terminated and resources as well as power are saved on the satellite.

The developer can choose whichever mechanism he prefers to pass around data between different applications. One could define a shared storage mount point, where MAIN saves all images before launching the CLOUD app with access to them and process them directly with file operations. Or, given that TCP/IP networking is feasible between the applications with hostname resolution, they can choose to use some RPC framework to pass around the required data. Given that gRPC and protos are already defined by the SCFW for the sensors of the satellite, it is simpler and faster to use the same message types and RPC framework to perform this.

To make things better, the CLOUD app can be reused for a new scenario that detects boats, without having to deploy a new container image from scratch.

```
#!/usr/bin/env python3

import os
import grpc
from proto.unibap_spacecloud_api_pb2 import RunApplicationRequest
from proto.unibap_spacecloud_api_pb2_grpc import ARMSStub


if __name__ == "__main__":
    print("Started Application")
    arms_channel = grpc.insecure_channel('172.17.0.1:50001')
    arms_stub = ARMSStub(arms_channel)
    cloud_hostname = os.getenv("CLOUD_HOSTNAME")
    try:

arms_stub.RunApplication(RunApplicationRequest(hostname=cloud_hostname))
    except grpc.RpcError as e:
        print(e)
    print("Application completed")
```

Code 18 Run concurrent application python

You can pull this container image at spacecloud.unibap.com/unibap/arms-demo:run-app

4.9 Releasing new applications

There are multiple ways a developer can release new applications. Docker allows to either upload a container image to a compatible docker registry or export the container image to a TAR (Tape Archive) file.

The SCFW has a private registry that SCFW users can upload their developed applications for testing on the SCHW dedicated testing systems.

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

4.9.1 Tagging the application

Before uploading an application, it is required to name the container image. You can perform this with a docker command on the terminal

```
docker tag {HASH_ID} {NAME:VERSION}
```

Code 19 Docker tag

Where {HASH_ID} is the 12 letter ID that docker build returns and {NAME:VERSION} is the path and version that should be uploaded. If you are uploading the image to the SCFW official registry, it must be of the form spacecloud.unibap.com/{ACCOUNT_NAME}/{APPLICATION_NAME}:{VERSION}.

4.9.2 Uploading the application

Before uploading the application, you must choose if you want that container image to be shared to other team members or even the entire SCFW community. In the SCFW registry website you can create a new organization, add new members to your organization and assign permissions to them.

Once you have performed these steps, make sure that the tag you have given to your container image is correct, if you wish to push the image in an organization, the tag must be of the form spacecloud.unibap.com/{ORGANIZATION_NAME}/{APPLICATION_NAME}:{VERSION}. To upload the image, simply type in a terminal

```
docker push {IMAGE_NAME:VERSION}
```

Code 20 Docker push

4.10 Advanced Usage

Dockerfiles can be used to create more complex deployment scenarios that can be used to both shrink the size of the developed application and perform more complex tasks such as code obfuscation.

The following example shows how one can use multistage building to create a much smaller container image that contains only the runtime dependencies of the app.

```
FROM spacecloud.unibap.com/unibap/framework-sdk AS builder
WORKDIR /app
RUN apt-get update -qq \
    && apt-get -qq install --no-install-recommends -y git
COPY my_cmake_project/* .
RUN mkdir build \
    && cd build \
    && cmake .. \
    && make

FROM spacecloud.unibap.com/unibap/framework-runtime
WORKDIR /app
COPY --from=builder build/app .
CMD ["./app"]
```

Code 21 Multistage building

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

5 Application Testing

Interaction with the SCFW, as described in section 3, is performed through the SCFW Administration. However, at this early stage, access to the SCFW Administration is currently limited to Unibap AB employees only. This means that, although developers can download the SDK and push new applications to the SCFW registry for testing, a manual process is required to run the developed applications in the dedicated SCFW Nodes.

To request for application testing on the system, one must send an e-mail to the spacecloud@unibap.com address with subject "SCFW Test" containing the following

- Execution parameters
- TLE for simulated orbit
- Dataset for image queries

5.1 Execution parameters

In the following YAML definition, the execution context is defined by two applications, test and sensors. The test app will get launched automatically when the context is executed (defined by the when key), while the sensors app will not get launched unless a RunApplicationRequest for it is sent to ARMS.

Similarly, test defines what command is to be executed when it is launched, while sensors does not define a command – meaning that the container image's default CMD command will be executed. Environment defines a list of environment variables that will be present when the application is executed, and storage defines the persistent storage spaces available. A volume name, mount point and mode (one of "rw" or "ro" for read-write and read-only) are required for each storage space. The resources section defines all the accessible hardware for the application, CPU cores, priority, memory, GPU and even VPU access. Logs are also defined in the last section of each application.

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

```yaml
applications:
  - image: spacecloud.unibap.com/unibap/hello-world
    hostname: test
    command: /bin/bash -c "touch /mnt/test && sleep 10"
    environment:
      - STORAGE: /mnt
    storage:
    - volume: volume_1
      mount: /mnt
      mode: rw
    resources:
      cpu:
        priority: 1024
        cores: 0.5
      memory:
        swap: 512m
        limit: 256m
        kernel: 32m
      gpu:
        enable: false
      vpu:
        enable: false
    logs:
      max-size: 10m
      max-file: 3
      compress: true
    when: auto
  - image: spacecloud.unibap.com/unibap/sensors-demo:simple
    hostname: sensors
    command: ~
    environment: []
    storage: []
    resources:
      cpu:
        priority: 1024
        cores: 0.5
      memory:
        swap: 512m
        limit: 256m
        kernel: 32m
      gpu:
        enable: false
      vpu:
        enable: false
    logs:
      max-size: 10m
      max-file: 3
      compress: true
    when: manual
```

Code 22 Execution parameters

5.2 Execution results

After the test run has been completed, you will receive the following e-mail

- Resource usage report
- All contents of the persistent storages that are marked as DOWNLINK

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).

- All logs gathered during execution of your applications
- Logs of all the API calls that your applications made to the SCFW ARMS and Sensors

5.3 Conformance testing

The SCFW Conformance Testing suite is part of SCFW Administration and allows all applications to be rigorously tested before being deployed to SCFW Nodes. This allows interested parties to test applications against different scenarios and policies to make sure that, once deployed, these applications follow all the regulations and standards defined by the administrators.

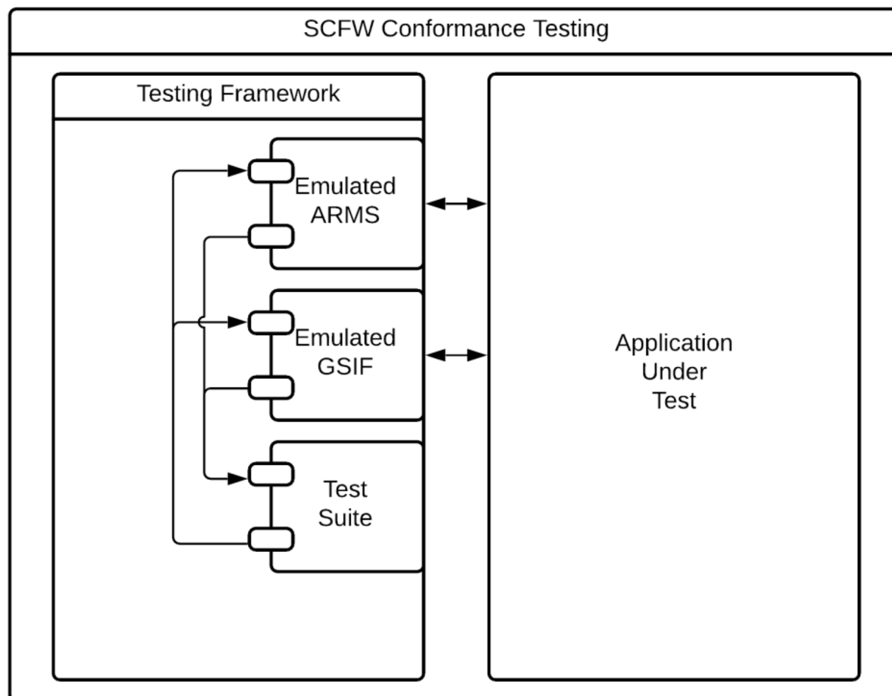

Figure 5 The SCFW Conformance Testing suite

Copyright © Unibap AB (publ.). All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of the information contained herein for other purposes than here intended, is not permitted except with the prior and written permission of Unibap AB (publ).