xApp descriptor

Obsolete Note

We are in the process of moving this xApp writing guide to https://docs.o-ran-sc.org.

Please refer to App Writing Guide for latest guide.

The xApp descriptor is provided by the xApp writer. xApp Descriptor includes all the basic and essential information for the RIC platform to manage the life cycle of the xApp. Information and configuration included in the xApp descriptor will be used to generate the xApp helm charts and define the data flows to the north and south bound traffics. xApp developer can also include self-defined internal parameters that will be consumed by the xApp in the xApp descriptor.

The xApp descriptor comes with its JSON schema file that validates it. Please see the schema section about the details.

Structure:

The xApp descriptor follows a JSON structure. The following are the key sections that defines an xApp.

- **xapp_name**: (REQUIRED) this is the unique identifier to address an xApp. A valid xApp descriptor must includes the xapp_name attribute. The following is an example.

```
"xapp_name": "example_xapp"
```

- **version**: (REQUIRED) this is the semantic version number of the xApp descriptor. It defines the version numbers of the xApp artifacts (e.g., xApp helm charts) that will be generated from the xApp descriptor. Together with the xapp_name, they defines the unique identifier of an xApp artifact that can be on-boarded, distributed and deployed. The following is an example.

```
"version": "1.0.0"
```

- **containers**: (REQUIRED) This section defines a list of containers that the xApp will run. For each container, a structure that defines the container name, image registry, image name, image tag, and the command that it runs is defined. The name and images are REQUIRED. The command field is optional. The following is an example that defines two containers.

```
"containers": [
    {
        "name": "example_container_1",
        "image": {
            "registry": "example_image_registry_1",
            "name": "example_image_name_1",
            "tag": "example_image_tag_1"
        },
        "command": "example_command_1"
    },
    {
        "name": "example_container_2",
        "image": {
            "registry": "example_image_registry_2",
            "name": "example_image_name_2",
            "tag": "example_image_tag_2"
        }
    }
]
```

- **controls**: (OPTIONAL) The control section holds the internal configuration of the xApp. Therefore, this section is xApp specific. This section can include arbitrary number of xApp defined parameters. The xApp consumes the parameters by reading the xApp descriptor file that will be injected into the container as a JSON file. An environment variable XAPP_DESCRIPTOR_PATH will point to the directory where the JSON file is mounted inside the container. If the controls section is not empty, the xApp developer must provide the schema file for the controls section. Please refer to Schema for xApp Descriptor for creating such schema file. The following is an example for the controls section.

```
    "controls": {
        "active": True,
        "requestorId": 66,
        "ranFunctionId": 1,
        "ricActionId": 0,
        "interfaceId": {
            "globalENBId": {
                "plmnId": "310150",
                "eNBId": 202251
            }
        }
    }
```

- **metrics**: (OPTIONAL) The metrics section of the xApp descriptor holds information about metrics provided by the xApp. Each metrics item requires the \textit{objectName}, \textit{objectInstance}, \textit{name}, \textit{type} and \textit{description} of each counter. The metrics section is required by VESPA manager (RIC platform component) and the actual metrics data are exposed to external servers via Prometheus/VESPA interface. The following is an example.

```
    "metrics": [
        {
            "objectName": "UEEventStreamingCounters",
            "objectInstance": "SgNBAdditionRequest",
            "name": "SgNBAdditionRequest",
            "type": "counter",
            "description": "The total number of SG addition request events processed"
        },
        {
            "objectName": "UEEventStreamingCounters",
            "objectInstance": "SgNBAdditionRequestAcknowledge",
            "name": "SgNBAdditionRequestAcknowledge",
            "type": "counter",
            "description": "The total number of SG addition request acknowledge events processed"
        }
    ]
```

- **messaging:** (OPTIONAL) This section defines the communication ports for each containers. It may define list of RX and TX message types, and the A1 policies for RMR communications implemented by this xApp. Each defined port will creates a K8S service port that are mapped to the container at the same port number. This section requires ports that contains the port name, port number, which container it is for. For RMR port, it also requires tx and rx message types, and A1 policy list.

Stop gap solution for bronze release

The messaging section replaces the previously RMR section in the xApp descriptor. It requires appmgr to modify its codes to parse the new messaging section. Before the new version of appmgr is released, as a stop gap solution, we will also include a compatible RMR section with the same information in the xApp descriptor. Please refer to the stop-gap-MCxApp descriptor for example.

choosing port numbers

In the bronze release appmgr is not consuming the port name defined in the messaging section yet. Please chose to use the default 4560 port for rmr-data and 4561 for rmr-route.

port naming convention

Kubernetes requires the port name to be DNS compatible. Therefore, please choose a port name that contains only alphabetical characters (A-Z), numeric characters (0-9), the minus sign (-), and the period (.). Period characters are allowed only when they are used to delimit the components of domain style names.

```
        "messaging": {
                "ports": [
                        {
                                "name": "http",
                                "container": "mcxapp",
                                "port": 8080,
                                "description": "http service"
                        },
                        {
                                "name": "rmr-data",
                                "container": "mcxapp",
                                "port": 4560,
                                "txMessages":
                                [
                                        "RIC_SUB_REQ",
                                        "RIC_SUB_DEL_REQ"
                                ],
                                "rxMessages":
                                [
                                        "RIC_SUB_RESP",
                                        "RIC_SUB_FAILURE",
                                        "RIC_SUB_DEL_RESP",
                                        "RIC_INDICATION"
                                ],
                                "policies": [1,2],
                                "description": "rmr data port for mcxapp"
                        },
                        {
                                "name": "rmr-route",
                                "container": "mcxapp",
                                "port": 4561,
                                "description": "rmr route port for mcxapp"
                        }
                ]
        },
```

- **liveness probes**: (OPTIONAL) The liveness probe section defines how liveness probe is defined in the xApp helm charts. You can provide ether a command or a http helm liveness probe definition in JSON format. This section requires initialDelaySeconds, periodSeconds, and either httpGet or exec.The following is an example for http-based liveness probes.

```
    "livenessProbe": {
        "httpGet": {
            "path": "ric/v1/health/alive",
            "port": "8080"
        },
        "initialDelaySeconds": "5",
        "periodSeconds": "15"
    },
```

  The following is an example for rmr-based liveness probes.

```
                "livenessProbe": {
                        "exec": {
                                "command": ["/usr/local/bin/rmr_probe"]
                    },
                    "initialDelaySeconds": "5",
                    "periodSeconds": "15"
                },
```

- **readiness probes:** (OPTIONAL) The readiness probe section defines how readiness probe is defined in the xApp helm charts. You can provide ether a command or a http helm readiness probe definition in JSON format. This section requires initialDelaySeconds, periodSeconds, and either httpGet or exec.The following is an example for http-based readiness probes.

```
    "readinessProbe": {
        "httpGet": {
            "path": "ric/v1/health/alive",
            "port": "8080"
        },
        "initialDelaySeconds": "5",
        "periodSeconds": "15"
    },
```

The following is an example for rmr-based readiness probes.

```

                "readinessProbe": {
                        "exec": {
                                "command": ["/usr/local/bin/rmr_probe"]
                    },
                    "initialDelaySeconds": "5",
                    "periodSeconds": "15"
                },
```

xApp Descriptor Example:

You can download the xApp descriptor for MC xApp here config-file.json

You can find the corresponding schema from here schema.json

Stop-gap xApp Descriptor Example for Bronze Release:

You can download the stop-gap xApp descriptor for MC xApp here stop-gap-...file.json.

The schema file is the same as the previous one.