**Container Networking Interface Specification**

# 概述

这篇文档为Linux上的容器提出了一种通用的基于插件的网络解决方法*CNI*.

它来源于[rkt Networking Proposal](https://docs.google.com/a/coreos.com/document/d/1PUeV68q9muEmkHmRuW10HQ6cHgd4819_67pIxDRVNlM/edit#heading=h.ievko3xsjwxd), 它致力于满足rkt网络中的许多设计思路.

为了本提案的目的，我们定义了２个非常明确的术语:

* 容器可以看作 [Linux *network namespace*](http://man7.org/linux/man-pages/man7/namespaces.7.html)的同义语. What unit this corresponds to depends on a particular container runtime implementation: for example, in implementations of the [App Container Spec](https://github.com/appc/spec) like rkt, each *pod*runs in a unique network namespace. In [Docker](https://docker.com/), on the other hand, network namespaces generally exist for each separate Docker container.
* *network* refers to a group of entities that are uniquely addressable that can communicate amongst each other. This could be either an individual container (as specified above), a machine, or some other network device (e.g. a router). Containers can be conceptually *added to* or *removed from* one or more networks.

这篇文档的目的是定义This document aims to specify the interface between "runtimes" and "plugins". Whilst there are certain well known fields, runtimes may wish to pass additional information to plugins. These extentions are not part of this specification but are documented as [conventions](https://github.com/containernetworking/cni/blob/master/CONVENTIONS.md). The key words "must", "must not", "required", "shall", "shall not", "should", "should not", "recommended", "may" and "optional" are used as specified in [RFC 2119](https://www.ietf.org/rfc/rfc2119.txt).

# 总指南

* 容器运行时在调用任何插件之前必须为容器创建一个新的网络命名空间.
* 运行时必须决策容器应该属于哪些网络，对于每种网络需要调用哪些插件.
* 网络配置是JSON格式并且容易存储在文件中. 网络配置包括强制的字段如"name" "type" 和插件特定的字段. The network configuration allows for fields to change values between invocations. For this purpose there is an optional field "args" which must contain the varying information.
* The container runtime must add the container to each network by executing the corresponding plugins for each network sequentially.
* Upon completion of the container lifecycle, the runtime must execute the plugins in reverse order (relative to the order in which they were executed to add the container) to disconnect the container from the networks.
* The container runtime must not invoke parallel operations for the same container, but is allowed to invoke parallel operations for different containers.
* The container runtime must order ADD and DEL operations for a container, such that ADD is always followed by a corresponding DEL. DEL may be followed by additional DELs, however, and plugins should handle multiple DELs permissively (i.e. plugin DEL should be idempotent).
* A container must be uniquely identified by a ContainerID. Plugins that store state should do so using a primary key of (network name, container id).
* A runtime must not call ADD twice (without a corresponding DEL) for the same (network name, container id). In other words, a given container ID must be added to a specific network exactly once.

# CNI插件

## 概述

每个CNI插件必须实现为一个可执行程序，它会被容器管理系统(如rkt或者k8s)调用.

一个CNI插件负责为容器的网络空间插件一个网络接口（如一个veth pair的一端） 并在host中做出必要的动作 (比如将veth的一端连接到一个网桥上). 插件应该调用合适的IPAM插件.

## 参数

CNI插件必须支持的操作:

* 将容器加入网络
  + 参数:
    - **Version**. 插件调用者使用的CNI spec版本(如容器管理系统).
    - **Container ID**. 运行时为容器分配的唯一标识，必须不为空.
    - **Network namespace path**. 代表将要加入的网络空间的路径.比如/proc/[pid]/ns/net或者指向它的bind-mount/link.
    - **Network configuration**. 用来描述容器加入的网络的JSON文档. 下面会描述文档结构.
    - **Extra arguments**. 这为CNI插件为不同的容器设置不同的配置提供了一个可选的机制.
    - **Name of the interface inside the container**. 为容器分配的接口名；必须满足基本的Linux接口名限制..
  + Result:
    - **Interfaces list**. Depending on the plugin, this can include the sandbox (eg, container or hypervisor) interface name and/or the host interface name, the hardware addresses of each interface, and details about the sandbox (if any) the interface is in.
    - **IP configuration assigned to each interface**. The IPv4 and/or IPv6 addresses, gateways, and routes assigned to sandbox and/or host interfaces.
    - **DNS information**. Dictionary that includes DNS information for nameservers, domain, search domains and options.
* 将容器从网络中删除
  + 参数:
    - **Version**. The version of CNI spec that the caller is using (container management system or the invoking plugin).
    - **Container ID**, as defined above.
    - **Network namespace path**, as defined above.
    - **Network configuration**, as defined above.
    - **Extra arguments**, as defined above.
    - **Name of the interface inside the container**, as defined above.
  + All parameters should be the same as those passed to the corresponding add operation.
  + A delete operation should release all resources held by the supplied containerid in the configured network.
* Report version
  + Parameters: NONE.
  + Result: 插件支持的CNI spec版本
  + {
  + "cniVersion": "0.3.1", // the version of the CNI spec in use for this output
  + "supportedVersions": [ "0.1.0", "0.2.0", "0.3.0", "0.3.1" ] // the list of CNI spec versions that this plugin supports

}

运行时必须使用网络的类型作为执行的插件名称(参考下面的网络配置).

运行时需要在一系列预定义的目录中查找要执行的插件。查找插件后，需要使用下面的环境变量来执行插件:

* CNI\_COMMAND: 指明要执行的操作; ADD, DEL or VERSION.
* CNI\_CONTAINERID: 容器IDContainer ID
* CNI\_NETNS: 网络空间文件路径
* CNI\_IFNAME:将要设置的接口名; 如果插件不能够使用这个接口名必须返回错误
* CNI\_ARGS: 执行插件时传递的其它参数. 以分号分隔的k-v对; 比如, "FOO=BAR;ABC=123"
* CNI\_PATH:搜寻CNI插件的路径列表.路径以特定的OS列表分隔符; 比如 ':' on Linux and ';' on Windows

JSON格式的网络配置必须通过插件的标准输入输入.这意味着不与特定的磁盘文件绑定并且可能在不同的插件间改变信息

## Result

注意IPAM插件应该返回一个在 [IP Allocation](https://github.com/containernetworking/cni/blob/master/SPEC.md#ip-allocation)中描述的缩略结构。

对于ADD操作插件必须返回0来表示成功和打印下在的JSON结果到标准输出.

ips和dns条目必须和IPAM插件的输出一致。(see [IP Allocation](https://github.com/containernetworking/cni/blob/master/SPEC.md#ip-allocation) for details) 插件需要恰当地填充 interface indexes, IPAM插件不会关心接口也不返回这些信息.

{

"cniVersion": "0.3.1",

"interfaces": [ (this key omitted by IPAM plugins)

{

"name": "<name>",

"mac": "<MAC address>", (required if L2 addresses are meaningful)

"sandbox": "<netns path or hypervisor identifier>" (required for container/hypervisor interfaces, empty/omitted for host interfaces)

}

],

"ips": [

{

"version": "<4-or-6>",

"address": "<ip-and-prefix-in-CIDR>",

"gateway": "<ip-address-of-the-gateway>", (optional)

"interface": <numeric index into 'interfaces' list>

},

...

],

"routes": [ (optional)

{

"dst": "<ip-and-prefix-in-cidr>",

"gw": "<ip-of-next-hop>" (optional)

},

...

]

"dns": {

"nameservers": <list-of-nameservers> (optional)

"domain": <name-of-local-domain> (optional)

"search": <list-of-additional-search-domains> (optional)

"options": <list-of-options> (optional)

}

}

.

cniVersion指定插件使用的CNI specification版本.

 interfaces描述了插件创建的网络接口.

如果CNI\_IFNAME变量存储，插件必须使用这个名字作为sandbox/hypervisor接口 or 或者返回错误.

* mac (string): 接口的硬件地址.如果L2地址对插件没有意义则字段是可选的.
* sandbox (string): 在container和基于namespace的环境中需要返回sandbox的网络空间的全路径. 基于Hypervisor/VM的插件需要返回接口将要加入的虚拟sandbox的唯一ID. 这个字段必须提供对于接口创建或者加入.

ips字段是一个IP配置信息列表. See the [IP well-known structure](https://github.com/containernetworking/cni/blob/master/SPEC.md#ips) section for more information.

 dns字段包含了通用DNS信息的字典. See the [DNS well-known structure](https://github.com/containernetworking/cni/blob/master/SPEC.md#dns) section for more information.

规范没有规定CNI插件如何处理这些消息。比如，可以生成/etc/resolv.conf文件注入容器的文件系统或者在host上运行一个DNS转发.

必须返回一个非０值来指示错误并向标准输出打印JSON信息:

{

"cniVersion": "0.3.1",

"code": <numeric-error-code>,

"msg": <short-error-message>,

"details": <long-error-message> (optional)

}

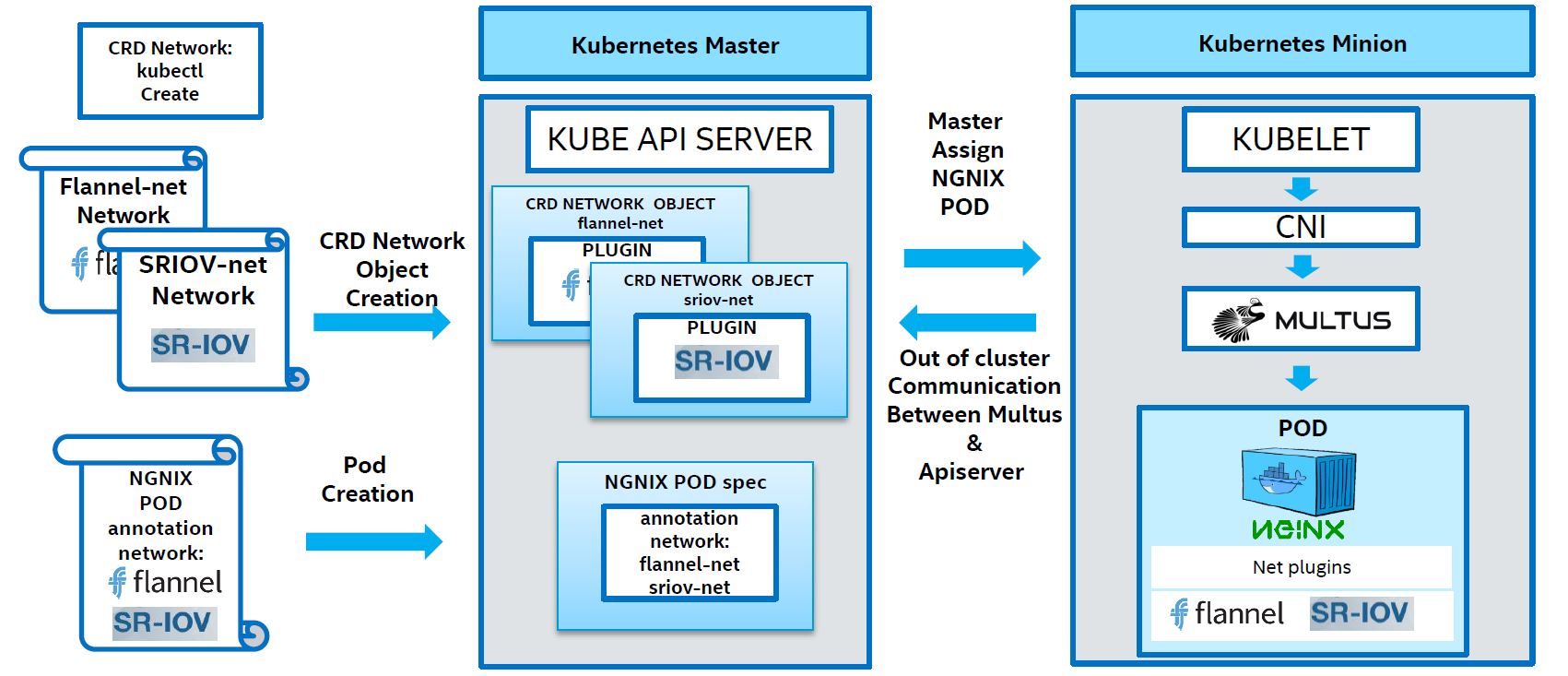
cniVersion指出了插件使用的CNI specification版本. Error codes 0-99 保留为通用错误 (see [Well-known Error Codes](https://github.com/containernetworking/cni/blob/master/SPEC.md#well-known-error-codes) section). 100+ 以上的错误码可以被插件使用.

另外, stderr可以用来输出非结构化信息比如logs.

# CNI-插件

## Multus

Kubelet负责为每个pod建立网络接口; 它通过执行配置的CNI插件来实现. 当执行multus时,它恢复Multus相关的pod注释, 反过来，它使用这些注释来恢复k8s的CRD, CRD指明了执行哪些插件和插件所使用的配置. 插件的执行顺序很重要，因为要区分主插件.



## 配置k8s

1. 创建第三方资源”crdnetwork.yaml”

apiVersion: apiextensions.k8s.io/v1beta1

kind: CustomResourceDefinition

metadata:

# name must match the spec fields below, and be in the form: <plural>.<group>

name: networks.kubernetes.com

spec:

# group name to use for REST API: /apis/<group>/<version>

group: kubernetes.com

# version name to use for REST API: /apis/<group>/<version>

version: v1

# either Namespaced or Cluster

scope: Namespaced

names:

# plural name to be used in the URL: /apis/<group>/<version>/<plural>

plural: networks

# singular name to be used as an alias on the CLI and for display

singular: network

# kind is normally the CamelCased singular type. Your resource manifests use this.

kind: Network

# shortNames allow shorter string to match your resource on the CLI

shortNames:

- net

1. 运行kubectl create命令创建CRD

# kubectl create -f ./crdnetwork.yaml

customresourcedefinition "network.kubernetes.com" created

1. kubectl get确认创建

# kubectl get CustomResourceDefinition

NAME KIND

networks.kubernetes.com CustomResourceDefinition.v1beta1.apiextensions.k8s.io

1. 创建flannel-network.yaml

apiVersion: "kubernetes.com/v1"

kind: Network

metadata:

name: flannel-networkobj

plugin: flannel

args: '[

{

"delegate": {

"isDefaultGateway": true

}

}

]'

1. 创建CRD

# kubectl create -f customCRD/flannel-network.yaml

network "flannel-networkobj" created

# kubectl get network

NAME KIND ARGS PLUGIN

flannel-networkobj Network.v1.kubernetes.com [ { "delegate": { "isDefaultGateway": true } } ] flannel

1. 获取CRD详细信息

# kubectl get network flannel-networkobj -o yaml

apiVersion: kubernetes.com/v1

args: '[ { "delegate": { "isDefaultGateway": true } } ]'

kind: Network

metadata:

clusterName: ""

creationTimestamp: 2017-07-11T21:46:52Z

deletionGracePeriodSeconds: null

deletionTimestamp: null

name: flannel-networkobj

namespace: default

resourceVersion: "6848829"

selfLink: /apis/kubernetes.com/v1/namespaces/default/networks/flannel-networkobj

uid: 7311c965-6682-11e7-b0b9-408d5c537d27

plugin: flannel

TPR和CRD有相同的链接 : **/apis/kubernetes.com/v1/namespaces/default/networks/**

## 配置multus

创建/etc/cni/net.d/multus-cni.conf配置文件。

{

"name": "minion-cni-network",

"type": "multus",

"kubeconfig": "/etc/kubernetes/node-kubeconfig.yaml"

}

重启kubelet

# systemctl restart kubelet

## 配置multus使用kubeconfig并使用默认网络

{

"name": "minion-cni-network",

"type": "multus",

"kubeconfig": "/etc/kubernetes/node-kubeconfig.yaml",

"delegates": [{

"type": "weave-net",

"hairpinMode": true,

"masterplugin": true

}]

}

重启kubelet

## 配置POD使用CRD网络对象

# cat pod-multi-network.yaml

apiVersion: v1

kind: Pod

metadata:

name: multus-multi-net-poc

annotations:

networks: '[

{ "name": "flannel-conf" },

{ "name": "sriov-conf"},

{ "name": "sriov-vlanid-l2enable-conf" }

]'

spec: # specification of the pod's contents

containers:

- name: multus-multi-net-poc

image: "busybox"

command: ["top"]

stdin: true

tty: true

# kubectl create -f ./pod-multi-network.yaml

pod "multus-multi-net-poc" created

# kubectl get pods

NAME READY STATUS RESTARTS AGE

multus-multi-net-poc 1/1 Running 0 30s