:Containter Orchestration

==========

- شكل الـ App في الـ Legacy Server .. كذا مشكلة منها low utilization of resources بسبب ان منقدرش نحط اكتر من app لهم dependecies مختلفة عن بعض .. وقت عشان اقوم physical server بالتالي low scalability.
- نقلنا لـ vm و containers .. مش بحجز resources في الـ container عشان هو في نظر الـ kernel حيكون process بتاخد الـ resources اللي محتاجها.
- كمان الـ os بيكون booted مرة واحدة في الـ container .. لإنه هي process بتيجي لها signal انها تشتغل.
- كمان الـ automation في الـ container وفر لي مبدأ الـ CI/CD اللي بقى لا غنى عنه في عالم الـ microservices .. محتاج fast scaling و الـ fast booting time.
- محتاج الـ Life Cycle Management .. محتاج tool تتأكد إن الـ containers are healthy and running .. محتاج كمان tool تكون مسؤولة عن الـ Containers Scheduling .. لو في ماشين خلاص بقت full utilized .. فلو في container محتاج يـ scheduled فيروح لماشين less utilized .. برضه تكون مسؤولة عن الـ Horizontal / Veritcal Scaling .. كمان Health check Monitor للـ Apps ... مش مجرد Start / Stop للـ Containers .. لو الـ Pod محتاجة gpu node .. فالـ tool هي المسؤولة عن الـ required node assignment.
- كمان الـ k8s مش بيـ manage pods فقط ... كمان بيـ manage nodes .. لو عندي node عاوز اخرجها من الـ cluster لـ failure أ, maintence مثلاً ... فـ k8s مسؤول عن الـ running pods on current node to the healthy available nodes ... عن طريق الـ Kublete اللي بيـ report الـ node health/status باستمرار للـ API Server.
- بعد ما الـ node بترجع بتكون monitored by k8s وبيتعمل عليها some checks ولما تـ passed .. بتبدأ في استقبال / scheduling الـ Pods.

:K8S Features

==========

01- Services Discovery: بتقوم مقام الـ stable fronend لكل الـ pods اللي نقدر نقول عليها انها - not stable backend- على مستوى نفس الـ application.

02- Load Balancing: لو عاوز اعمل expose الـ service بره الـ cluster عشان تكون managed by a dedicated load balancer. بتعريف الـ ClusterIP والـ NodePort. ودول كده نوعين من الـ Endpoints اللي تخلينا قادرين نتعامل مع الـ pods >>> لكن برضه k8s بيقدر يـ balance loads بشكل efficient وسريع جداً لان كل الـ traffic جوه الـ cluster .. بتشوف الـ pod الجاهز لاستقبال الـ traffic وتبدأ تبعته ليها.

<<<< دول كده جزء من الـ Service Object >>>>

03- Self-Healing: عن طريق طريقتين بيقدر يكون متوفر لنا الـ Reliability .. عن طريق طريقتين الـ Readiness probe and Lifeness Probe. الـ Readiness مش بس بعد ما الـ pod بتقوم .. لكن كمان على طول الـ Life Cycle >> طول ما الـ pod is ready فبتقدر تستقبل الـ traffic.

04- Application Hosting: الـ Developer بيكون Abstracted عن أي حاجة Infrastructure وبيكون تركيزه فقط على الـ Application.

Cluster Archeticture:

==========

- ايه مكوناته؟ .. k8s عبارة عن distributed platform بتتكون من Master as Server و Worker as Client ...
- الماستر هو اللي بيبعت الـ instructions للـ Worker nodes >>> بيتكون من API Server وده الـ Central Communication / Gateway اللي بتهندل الـ Requests بين باقي مكونات الـ master node أو حتى الـ worker node من نفس الـ cluster أو nodes from other cluster ،،، يجي بعد كده الـ Controll Manager المسؤول عن وصول الـ Current Cluster State للـ Desired Cluster State .. بيفضل يقارن بين الـ Current و الـ Desired ويتأكد انهم Matched. وكمان دايماً باستمرار بيكلم الـ API Server عشان لو في Task مطلوب تنفيذها يقوم بده.
- وعندنا كمان الـ Scheduler: هو المسؤول عن pod assignment for suitable node. عن طريق factors مختلفة عشان يختار أفضل node يحط الـ pod عليها.

بالنسبة للـ Worker Node:

- عندي أول حاجة الـ Kubelet ده نقدر نعتبره الـ agent اللي بيكون موجود وشغال في كل Worker Node وهو اللي بيسجلها عند الـ Master Node API Server وبيبلغ باستمرار الـ Worker Node Status ليه .. وكمان بتستنى منه أي instruction اللي المفروض تنفذها.

Create Object Process:

==========

1- أو حاجة بيبعت للـ Master Node API Server الـ Request اللي بيقول له فيه انه محتاج يعمل Desired State لـ Deployment ولتكن Nginx مثلاً.
2- الـ API Server بعد ما يستلم الطلب بيبدأ يعمل بعض الـ Checks اللي من خلالها بيـ Authenticate الـ Client الأول "logging" >> وبعد كده اشوف اذا كان الـ Request مسموح لليوزر انه يعمله ولا لأ .. إلخ.
3- يجي دور الـ ETCD بعد ما الـ Checks تكون Passed .. الـ API Server بيخزن الـ Deployment Object في الـ ETCD Database
4- الـ ETCD بدوره حيبلغ الـ API Server انه خزن الـ Object بنجاح.
5- الـ API Server بيبلغ الـ User إن الـ Object Created.
6- الـ Controller Manager زي باقي الـ Components بيـ listen على الـ API Server وحيلاقي تاسك / لوجيك إن مطلوب يكون في Deployment مطلوب انها تكون موجودة في الـ cluster وهي مش موجودة .. فالـ CM حيلاقي انه الـ Current State =! Desired State >>> فاللي حيحصل ان الـ CM حيعمل Create للـ Deployment وبعده الـ ReplicaSet Controller حيـ Create Pods بعدد حسب المطلوب منه في الـ Deployment Object >> كل الـ Objects دي بترجع باستمرار على الـ API Server و الـ API Server بيخزنها باستمار على الـ ETCD. الـ Pods اتعملت ولكن لسه not assigned على أي node >>> لسه الـ Status بتاعتها Pending.
7- الـ Scheduler زيه زي الـ CM وهو بيسمع الـ API Server .. فحيلاقي مبعوت له Nodes بـ Pending Status >>> فحينفذ اللوجيك بتاعه ويشوف الـ nodes ويبدأ يـ assign الـ pod object requirements ويختار له الـ node اللي تـ match them ... هنا الـ Sched مش مطلوب منه تشغيل الـ pod .. كل اللي مطلوب منه يـ upadate الـ node specs لكل pod في الـ ETCD عن طريق الـ API Server برضه.
8- الـ Kubelete باستمرار برضه سامعة الـ API Server .. فحتلاقي لوجيك مطلوب منها بإن في 3 pods مثلاً مطلوب 2 منهم يكونوا على 1 node و 1 منهم على 2 node فهنا الـ Kubelete حتشغل الـ pods ولكن مش هي اللي حتعمل الـ Container.

9- لكن الـ Container Runtime هي اللي حتستلم Request من الـ Kubelete وهي اللي حتعمل الـ Container وبكده الـ Deployment اتعملت والـ Pods اشتغلت.

Master Components:

==========

- الـ communication بين الـ API Server وبين أي Component بيتم عن طريق Certificate / متشفر زي الـ PKI.
- من ضمن الـ pod specs اللي بيحدد على اساسها الـ scheduler حيـ assign كل pod على انه node .. ان الـ pod محتاج affinity أو anit-affinity يعني هل مجموعة الـ pods شرط انها تشتغل مع بعض على نفس الـ node ولا العكس على nodes مختلفة.
- Etcd مش مقبول انها تكون down تحت اي ظرف لإن ده معناه ان الـ Cluster كله down. بالتالي من اهميتها انا محتاج اضمن ان يكون معايا backup باستمرار منها .. عشان اقدر restore state of cluster .. في كل master node بيكون معايا جواها etcd instance وده موديل بيكون اسمه stacked etcd و الـ api server بيكلم الـ instance ده. أو ممكن نجمع الـ etcd instanced في external cluster والـ api server بيتكلم معاها. ده افضل عشان اضمن High Availability.
- كام master node بيكون على حسب حجم الـ traffic من cluster traffic اللي بيجي على api server لو الـ user محتاج يعمل object أو يـ scale deployment يعمل service ...
- لكن الـ app traffic بيروح للـ ingress controller ومنه بيوجه الـ traffic ناحية الـ worker nodes وبالتالي ده محسوب معانا كـ cluster traffic .. لكن لو في update / scale request محتاج يعمله الـ app فساعتها ده مش حيتم لو الـ master node واقعين كلهم.
- كام etcd؟ ... يفضل يكون الحد الأدنى 3 والحد الأقصى مش مفتوح .. لسبب إن الـ writing بيكون sequential بسبب ان مسموح لـ leader etcd node فقط والـ follower بيـ sync data كل واحد مع اللي قبلها بشكل sequencial as chain فكده العدد الكبير منهم حيسبب latency عقبال ما الـ update يحصل على كل الـ etcd nodes.
- بالنسبة للـ Request Validation اللي بيجي من الـ user: هل الـ user له access / authenticated على الـ cluster .. لو ليه فحيشوف لو هو authorized / has permission انه يعمل له الـ request ... بعد كده بتيجي مرحلة Mutating Admission Controller ... هنا بيتغير شكل الـ object تحسباً لو في bug في الـ object specs أو حاجة الـ user مش محدد تقدر تخلي الـ pod بشكلها الحالي تاخد كل الـ resources اللي في الـ cluster .. في المثال ده المسؤول عن تفادي ان ده يحصل حاجة اسمها limit range admission controller.
- بعد كده تيجي مرحلة الـ Schema Validation ... بيكون للـ object بعد ما اتعدل من الـ M A C قبل ما يسجله في الـ etcd database.
- بتيجي بعد كده اخر مرحلة Validation Admission Controller ... هنا ده final validation اني بحط policy زي ان اليوزر مش مسموح له يـ deploy public image .. لازم تكون من الـ internal registry ... أو انه ممنوع من استخدام الـ latest tag ... أو الـ deployment تكون باستخدام deployment pipeline زي الـ Jenkins ... لازم تكون باستخدام authorized pipline.
- لما كل ده يتم ... نقدر نحط الـ object في الـ etcd database.

Worker Components:

==========

- Kubelet تعتبر الـ agent اللي بتكون موجودة في كل node ودي بتكون مسؤولة عن تسجيل / register و تـ report the status الـ node في الـ API Server من readiness & liveness & utilization.
- بيكون في interface بين الـ kubelet وبين الـ Container Runtime "CRI" بتبعت فيه تعليمات وشكل الـ container المطلوب.
- Kube Proxy موجود في كل worker node ودوره هو service management .. عن طريق انه برضه بيشوف الـ API Server باستمرار .. وأول ما يلاقي ان في service اتعملت .. فهو اللي بيعمل لها rule بـ virtual ip خاص بيها في الـ ip tables ... نفس الكلام بالنسبة للـ pods لما بتتعمل لكنه مش بيهندلها هي بشكل مباشر .. بيكون الـ Endpoint controller manager هو اللي أول ما يلاقي pod/s اتعملت بيعمل لها حاجة اسمها Endpoint ... الفكرة اني ضامن ان ده ip قادر انه يستقبل traffic بالتالي يقدر يتعامل معاه الـ kube proxy ويعمل لها rule بالـ Service و الـ Endpoints.