CSC547 Homework Assignment #5

Akruti Sinha (@asinha6) || Priya Andurkar (@pandurk)

Problem 1:

5.2) Deployments. The following are some of the benefits of canary deployments. Please provide further context on how canary deployments enable the following:

- **A/B testing: ...**
- **Capacity Test: ...**
- **User Feedback: ...**
- **No cold-starts: ....**
- **No Downtime ...**
- **Easy Rollback ...**

Your answers should be minimally 2 sentences and not more than 3-4 sentences.

Answer:

1. A/B testing: Canary deployment provides controlled exposure of the new software to the users. It divides the users into two groups. The canary group uses the new version, and the control group uses the old stable version. This helps in comparing performance, user experience, and other metrics between new and old versions accomplishing the A/B testing.
2. Capacity test: This is achieved by gradually increasing the number of servers or users with new versions of the software. It helps in assessing whether the system is able to handle increased load, if it is scaling efficiently, and that the demand of the increased load is being met.
3. User Feedback: As this deployment strategy focuses on giving early access to a small subset of users, real-time valuable feedback can be gathered from their usage of the new features. These can be used to make improvements and fixes before rolling the new version out to the entire set of users.
4. No cold-starts: A cold-start or a dormant state of the application can sometimes impact latency and performance if there's a delay in initializing resources needed by the application. However, in canary deployments a subset of servers is already running the new version of the software. The application is warmed up before the main user base can use it and reduces the chances of witnessing any performance issues.
5. No downtime: the deployment downtime is minimized in case of canary deployments.This is achieved by gradually shifting traffic from the old version to the new version. The incremental approach of shifting traffic ensures that the majority of the users would get continued access to the application even if some changes are being deployed in the background.
6. Easy Rollback: In canary deployments, if issues are detected in the new version, it would be an easy change to redirect the traffic to the previous stable version because those

instances are still up and running. Recovery and rollback are straightforward because the majority requests are being routed to the older instances. In such an instance, the canary group would also get rerouted back to the old stable version.

Problem 2:

5.4) k8s concepts and terminology. Pod is a fundamental concept in k8s. The word pod as well as the acronym POD are used in several contexts.

1. **Find its precise, scientific meaning in Biology, Geology and Computing.**
2. **Repeat for its use in the Angling, Aviation and Cooking professions.**
3. **Expand the acronym POD in the fields of financing, sales, shipping, business, sales, and marketing.**
4. ~~**(Optional) We may use tripods for taking photographs; there are podiatrists in the medical profession. How is the word used in these contexts?**~~
5. ~~**(Optional) We listen to podcasts. How is the word used in these contexts?**~~
6. **(Last but not least) Search for synonyms of pod. Which one, in your opinion, comes closest to the use of the word in k8s? Why?**

Answer:

The term "pod" is used in a variety of contexts, and its exact meaning varies depending on the topic. Here's an examination of its meaning in different contexts.

1. **Biology**: A pod is a dried fruit or seed shell that opens along one or two sutures to release its seeds in biology. Pea pods, bean pods, and milkweed pods are all examples of pods. It could also mean - a group of related creatures that live in close proximity. Dolphin pods, whale pods, and locust pods are all examples of pods.
   Geology: A pod is a small, lenticular body of rock that is encased in another type of rock in geology. Pods can arise as a result of a number of geological processes, including igneous intrusion, metamorphic crystallization, and sedimentary deposition.
   Computing: A "pod" is a key unit of deployment in the context of Kubernetes (K8s). In the Kubernetes object model, it is the smallest deployable object. A pod can hold one or more containers that share the same network namespace and can easily communicate with one another. In Kubernetes, pods are used to execute application instances and are designed to be ephemeral, which means they can be easily produced, scaled, and removed.
2. **Angling**: A "pod" in angling can refer to a rod pod, which is a piece of equipment that fisherman use to support many fishing rods. The pod organizes and positions the rods for fishing.
   Aviation:A "pod" in aviation can refer to a streamlined exterior structure or container attached to the fuselage of an aircraft. Depending on the aircraft's objective, these pods can house a variety of equipment, such as sensors, cameras, or additional fuel tanks.
   Cooking: A "pod" is not a regularly used term in cooking with a definite scientific

meaning. It's worth mentioning, however, that the term "pod" can also be used informally to refer to a garlic pod or a chili pod, denoting a single clove of garlic or chili pepper.

3. In various business situations, the abbreviation "POD" can have multiple meanings.
    a. **Finance**: POD stands for Proof of Delivery. In the world of finance, "POD" might stand for "Proof of Delivery." It is a document or piece of evidence confirming the successful and complete delivery of goods or services to a customer. Proof of Delivery is required for invoicing, payment processing, and resolving any delivery-related disputes.
    b. **Sales**: POD stands for Point of Decision. - "POD" in sales might stand for "Point of Decision." The key time when a potential consumer is about to make a purchasing decision is referred to by this word. It is the stage at which sales and marketing efforts are aimed at influencing the customer's decision and persuading them to purchase a product or service.
    c. **Shipping**: POD stands for Proof of Delivery. In the transportation sector, "POD" can be an abbreviation for "Proof of Dispatch." It is documentation that proves that a shipment or item was delivered from the sender or seller to the recipient or customer. Tracking numbers, shipping labels, and other information may be included in Proof of Dispatch.
    d. **Business**: POD stands for Plan of Development. In the business world, "POD" might stand for "Plan of Development." This term is frequently used to describe a thorough and strategic strategy for a company's growth and expansion. It could include things like product development, market expansion, and resource allocation.
    e. **Sales** (Alternate): POD stands for Product Order Desk. In the sales world, "POD" can also stand for "Product Order Desk." This is a department or team within a firm that is in charge of processing and managing customer product orders. The Product Order Desk ensures that orders are logged, fulfilled, and delivered to customers accurately.
    f. **Marketing**: POD stands for Point of Difference. "POD" in marketing can stand for "Point of Differentiation." This refers to the distinct qualities, features, or characteristics that distinguish a product, service, or brand from competitors in the market. Identifying and emphasizing a strong point of differentiation is essential for effective marketing strategy.
4. (skip)
5. (skip)
6. A "pod" in the context of Kubernetes is effectively a container that runs one or more containers. In this context, a container is a lightweight, standalone executable package that contains everything required to run a piece of software, including the code, runtime, system tools, and libraries. In the Kubernetes ecosystem, the terms "container" and "pod" are frequently used interchangeably since they encapsulate the application and its dependencies, which is similar to the concept of a pod.

Problem 3:

5.10) k8s best practices. Find out any reference on k8s best practices. If your search came out empty, try https://www.densify.com/kubernetes-tools/kubernetes-best-practices/. It describes ten such practices.

1. **Which role(s) are these suggestions targeting?**
2. **Pick one suggested practice and expand on its pros and cons.**
3. **Find out the default value of the "liveness probe" frequency.**
4. **Find out the default values of the "Resource Requests and Limits" parameters.**

Answer:

1. These recommendations are aimed at developers and system administrators who are in charge of administering Kubernetes clusters. This can be shown by the best practices given below:
    a. Use the latest version of Kubernetes
    b. Use version control for configuration files
    c. Use namespaces to organize your cluster
    d. Use labels to organize your cluster resources
    e. Use readiness and liveness probes to ensure that your pods are healthy
    f. Use RBAC and a firewall to secure your Kubernetes cluster
    g. Set resource requests and limits for your pods
    h. Use smaller container images
    i. Audit your logs regularly
    j. Monitor your control plane components
2. Using readiness and liveness probes to ensure the health of your pods is one recommended approach. Readiness probes are used to determine whether or not a pod is prepared to receive traffic. Liveness probes are used to check whether a pod is still alive and, if not, whether it should be restarted.
    Pros:
    a. Improved reliability: Readiness and liveness probes can help to increase application dependability by ensuring that pods are only restarted when they are not healthy.
    b. Reduced downtime: Readiness and liveness probes can help to decrease downtime by minimizing unnecessary restarts of pods.
    c. Increased scalability: Readiness and liveness probes can help to increase the scalability of your applications by allowing you to scale them up and down without worrying about pods being restarted unnecessarily.

    Cons:

    d. Increased complexity: Because you must configure and maintain readiness and liveness probes, they might add complexity to your applications.

    e. Performance overhead: Because readiness and liveness probes must be conducted on a regular basis, they can introduce a minor performance overhead to your applications.
3. The official Kubernetes documentation states that the default value for the liveness probe frequency is 10 seconds.
4. From the listed reference [1], it can be inferred that there are no default values for "Resource Requests and Limits". The following could be the possible outcomes of not specifying Resource Limits and Resource Requests:
    a. When we don't specify the CPU limits, the CPU limit is decided on the basis of the upper bounds of the Node/Namespace in which the container is running.
        i. The Container may end up using all of the CPU resources available on the Node where it is running.
        ii. If namespace has a default CPU limit, then the Container running is automatically assigned the default limit.
    b. When the value for the Resource Requests is not specified, then Kubernetes assigns a value matching the Resource Limit.

References:

[1] https://kubernetes.io/docs/tasks/configure-pod-container/assign-cpu-resource/

Problem 4:

5.11) k8s HPA. Read the HPA walkthrough described in https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/. In order to run the experiments, you must select one of the three environments mentioned: minikube, killercoda or Play with Kubernetes. If you have access to an application other than the php-apache server, we'd prefer to use it instead.

1. **Run the experiments as described. Observe and report how HPA scales up and down.**
2. **Run experiments with different CPU target percentages, keeping load generation the same. Observe and report how HPA scales up and down. Explain any differences from the results of the previous question.**

Answer:

1. As per the experiments, we first create the Deployment and Service resources for the php-apache server. Once those are created, we create the HPA, which maintains the replicas between 1 to 10. The number of pods are increased when the CPU utilization exceeds 50% across all pods. Following is the behavior of horizontal scaling when load is generated:

```
priyaandurkar@Priyas-Air-4 ~ % kubectl get hpa php-apache --watch
NAME         REFERENCE                TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   0%/50%    1         10        1          5s
php-apache   Deployment/php-apache   104%/50%   1         10        1          46s
php-apache   Deployment/php-apache   104%/50%   1         10        3          61s
php-apache   Deployment/php-apache   64%/50%    1         10        3          106s
php-apache   Deployment/php-apache   84%/50%    1         10        3          2m46s
php-apache   Deployment/php-apache   84%/50%    1         10        6          3m1s
php-apache   Deployment/php-apache   43%/50%    1         10        6          3m46s
php-apache   Deployment/php-apache   29%/50%    1         10        6          4m46s
php-apache   Deployment/php-apache   0%/50%     1         10        6          5m47s
php-apache   Deployment/php-apache   0%/50%     1         10        6          8m32s
php-apache   Deployment/php-apache   0%/50%     1         10        4          8m47s
php-apache   Deployment/php-apache   0%/50%     1         10        4          10m
php-apache   Deployment/php-apache   0%/50%     1         10        1          10m
^C%
priyaandurkar@Priyas-Air-4 ~ %
```

From the above screenshot we can infer the following:

a. Initially the CPU consumption is 0% as there are no requests being sent.
b. Once the load generation begins, the CPU consumption becomes high and subsequently the HPA increases the number of replicas.
c. While the cpu consumption across all the pods controlled by the corresponding deployment are above 50%, the number of replicas keep increasing.
d. Once we stop the load generation process, the scaledown begins. This results in the removal of the replicas to the specified minimum value and coming back to 1 because there is no load.

During the scaling process, we can also see the number of replicas after scale up and scale down were as follows:

```
priyaandurkar@Priyas-Air-4 ~ % kubectl get deployment php-apache
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
php-apache   6/6     6            6           161m
priyaandurkar@Priyas-Air-4 ~ % kubectl get deployment php-apache
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
php-apache   1/1     1            1           171m
priyaandurkar@Priyas-Air-4 ~ %
```

2. We repeated this experiment by updating the CPU consumption target limit to 20% and 80%.

When the CPU limit for consumption is 20%, we observed the following:

```
priyaandurkar@Priyas-Air-4 ~ % kubectl get hpa php-apache --watch
NAME         REFERENCE                  TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   <unknown>/20%   1         10        0          6s
php-apache   Deployment/php-apache   0%/20%          1         10        1          6s
php-apache   Deployment/php-apache   215%/20%        1         10        1          66s
php-apache   Deployment/php-apache   215%/20%        1         10        4          81s
php-apache   Deployment/php-apache   215%/20%        1         10        8          97s
php-apache   Deployment/php-apache   215%/20%        1         10        10         112s
php-apache   Deployment/php-apache   42%/20%         1         10        10         2m7s
php-apache   Deployment/php-apache   25%/20%         1         10        10         3m7s
php-apache   Deployment/php-apache   9%/20%          1         10        10         4m7s
php-apache   Deployment/php-apache   0%/20%          1         10        10         5m7s
php-apache   Deployment/php-apache   0%/20%          1         10        10         8m52s
php-apache   Deployment/php-apache   0%/20%          1         10        5          9m7s
php-apache   Deployment/php-apache   0%/20%          1         10        5          9m52s
php-apache   Deployment/php-apache   0%/20%          1         10        1          10m
^C%
priyaandurkar@Priyas-Air-4 ~ %
```

Here, since the threshold for CPU consumption is significantly lesser than case 1, the number of replicas increases rapidly when we generate load as opposed to the pattern observed in case 1 as the consumption reaches the threshold much faster. We see that the maximum number of replica sets are reached to accommodate the CPU limit requirements. At this point, after stopping the load-generation the number of replicas are removed and return back to the set minimum of 1.

Similarly, when the CPU limit for consumption is 80%, we observed the following:

```
priyaandurkar@Priyas-Air-4 ~ % kubectl get hpa php-apache --watch
NAME         REFERENCE                  TARGETS         MINPODS   MAXPODS   REPLICAS   AGE
php-apache   Deployment/php-apache   <unknown>/80%   1         10        0          4s
php-apache   Deployment/php-apache   0%/80%          1         10        1          15s
php-apache   Deployment/php-apache   122%/80%        1         10        1          45s
php-apache   Deployment/php-apache   122%/80%        1         10        2          60s
php-apache   Deployment/php-apache   141%/80%        1         10        2          105s
php-apache   Deployment/php-apache   126%/80%        1         10        2          2m45s
php-apache   Deployment/php-apache   126%/80%        1         10        4          3m
php-apache   Deployment/php-apache   76%/80%         1         10        4          3m45s
php-apache   Deployment/php-apache   62%/80%         1         10        4          4m45s
php-apache   Deployment/php-apache   64%/80%         1         10        4          5m45s
php-apache   Deployment/php-apache   20%/80%         1         10        4          6m45s
php-apache   Deployment/php-apache   0%/80%          1         10        4          7m45s
php-apache   Deployment/php-apache   0%/80%          1         10        4          11m
php-apache   Deployment/php-apache   0%/80%          1         10        1          11m
php-apache   Deployment/php-apache   0%/80%          1         10        1          12m
^C%
priyaandurkar@Priyas-Air-4 ~ %
```

The threshold for CPU consumption in this case is significantly higher than case 1. The number of replicas increases at a slower rate in this case as the consumption percentage does not increase as much as case 1. Hence the maximum number of replicas which can handle the incoming load of requests goes up to 4. At this point, when we stop the load generation, downscaling occurs, and goes to the minimum of 1 replica.

The major difference observed with different CPU target percentages with the same load generation is that the maximum number of replicas created differs because the scale up is dependent on the CPU consumption target. In case when it is lesser, it will scale up to a higher number closer to the maximum. In case when the target is greater, it will scale to a number lower than the maximum number of replicas in case when the target is 50%. All of them respond appropriately to the burst of loads by scaling up as required. However, the maximum replicas may differ when we only change the CPU target and not the rate of load generation.

Bonus Problems:

Problem 6:

5.22) k8s and CI/CD. Handout H6.1 mentions some advantages k8s offers in the CI/CD use case.

1. Describe briefly what a CI/CD pipeline is.
2. How many advantages does the handout mention? Expand on two of them, using your own examples. Mention explicitly who benefits from these advantages.
3. Since there is no free lunch in engineering, there must be disadvantages. Identify at least one. Mention explicitly who "suffers" from these disadvantages.

Answer:

1. A CI/CD pipeline is a collection of automated steps used by software development teams to build, test, and deploy applications. Typical CI/CD pipelines include the following steps:
    a. Continuous integration (CI) is the process by which developers contribute their code to a source control repository such as Git. A CI server then builds and tests the code automatically, reporting any issues.
    b. Continuous delivery (CD): If the code passes all tests, the CI server can send it to production automatically. The team can also manually authorize the deployment.
2. The handout mentions advantages like automation, efficiency, portability. Expanding on two of them:
    a. **Automation**: By offering a framework for running and managing containerized build, test, and deployment processes, Kubernetes can assist teams in automating their CI/CD pipelines. Teams, for example, can use Kubernetes to run a continuous integration job that builds and tests their code with each commit. They can also utilize Kubernetes to execute a continuous delivery job that publishes their code to production as soon as all tests pass. The benefits are as follows:
        i. Instead of wasting time on manual deployment and administrative activities, developers may focus on writing code and fixing issues.
        ii. Operations teams can devote more time to strategic projects and less time to infrastructure management.

iii. Businesses can deliver new features and upgrades to clients more rapidly and consistently.
An example would be as follows: Kubernetes is used by a web application development company to automate its CI/CD pipeline. A CI task is automatically launched when a developer commits code to the source control repository. The code is built and tested by the CI task, and if all tests pass, it is deployed to production.

b. **Efficiency**: Kubernetes can help teams make better use of their resources by allowing them to operate numerous containers on the same physical server. This can decrease the number of servers that teams must manage while also saving money on cloud computing charges. The benefits are as follows:
   i. Infrastructure costs are being reduced.
   ii. Enhanced resource usage
   iii. Enhanced scalability

An example would be as follows: Kubernetes is used to run game servers by a company that makes mobile games. Kubernetes enables the company to scale up or down its gaming servers based on demand, ensuring that it can always fulfill the needs of its gamers.

3. Kubernetes has the disadvantage of being difficult to set up and administer. This can be difficult for teams that are unfamiliar with Kubernetes. Several people can 'suffer' from this disadvantage:
   a. DevOps teams that are new to Kubernetes may struggle to set up and manage Kubernetes clusters.
   b. Teams with limited resources may lack the knowledge or time to manage Kubernetes clusters.