When you create a new pod using `kubectl`, the request flows through several components within the Kubernetes architecture. Here's a step-by-step breakdown of how this process works:

**Step 1: kubectl Command**

Everything starts with you issuing a command using `kubectl`, the command-line interface for interacting with the Kubernetes cluster. For example, to create a new pod, you might use:

**kubectl run nginx --image=nginx**

This command tells Kubernetes to run a new pod with the `nginx` Docker image.

**Step 2: Authentication & Authorization**

When you issue the command, `kubectl` communicates with the API Server on the master node. The request is made via HTTPS, ensuring it's secure. The first thing the API Server does is authenticate the user, typically using certificates, bearer tokens, or other authentication methods.

Once authenticated, the request is then authorized. Kubernetes checks whether the user has the necessary permissions to perform the action, using roles and role bindings that define what actions users can perform on which resources.

**Step 3: API Server Validation**

Assuming the request is authorized, the API Server then validates the request to ensure it is correct and makes sense. This involves checking if the request is structurally sound and if the parameters provided (like the image name) are valid.

**Step 4: Storing in etcd**

Once the request is validated, the API Server writes the new desired state (to create a new pod) to the etcd datastore. etcd is a key-value store that Kubernetes uses to maintain all cluster data, ensuring that the cluster state is persisted and distributed.

**Step 5: Scheduler**

After the new pod is registered in etcd, the Kubernetes Scheduler becomes active. The Scheduler watches etcd for new pods that don’t yet have a node assigned. It selects a node for the pod to run on, based on resource availability, affinity specifications, constraints, and other policies. The decision of which node the pod should run on is then communicated back to the API Server, which updates the pod's information in etcd with the designated node.

**Step 6: Kubelet Starts the Pod**

Each node in the cluster runs an agent called the Kubelet. The Kubelet is responsible for ensuring that all containers described by PodSpecs are running and healthy. The Kubelet on the assigned node watches the API Server for pods that have been scheduled to its node. When it sees a new pod assigned to it, the Kubelet fetches the pod specification from the API Server and starts the container(s) specified in the pod using the container runtime (e.g., Docker, containerd).

**Step 7: Container Runtime**

The container runtime pulls the necessary image (in this case, `nginx`) if it is not already present on the node, and starts the container. Once the container is running, the Kubelet manages its lifecycle based on the pod specification.

**Step 8: Kubelet Updates the API Server**

After the pod starts running, the Kubelet updates the API Server with the status of the pod. This information is stored in etcd, allowing other components of the Kubernetes system to know the current state of the pod.

**Step 9: Kube-Proxy and Networking**

The Kube-Proxy on each node manages the network connectivity to the pods. This includes setting up network rules to ensure that the pod can communicate with other pods across the node network and potentially the outside world, depending on the network and security configurations.