|  |
| --- |
| package main |
|  |  |
|  | import ( |
|  | "fmt" |
|  | "sort" |
|  | "strconv" |
|  | "strings" |
|  | "testing" |
|  | "time" |
|  |  |
|  | "github.com/docker/docker/api/types/versions" |
|  | "github.com/docker/docker/integration-cli/cli" |
|  | "github.com/docker/docker/integration-cli/cli/build" |
|  | "github.com/docker/docker/pkg/stringid" |
|  | "gotest.tools/assert" |
|  | is "gotest.tools/assert/cmp" |
|  | "gotest.tools/icmd" |
|  | "gotest.tools/skip" |
|  | ) |
|  |  |
|  | func (s \*DockerSuite) TestPsListContainersBase(c \*testing.T) { |
|  | existingContainers := ExistingContainerIDs(c) |
|  |  |
|  | out := runSleepingContainer(c, "-d") |
|  | firstID := strings.TrimSpace(out) |
|  |  |
|  | out = runSleepingContainer(c, "-d") |
|  | secondID := strings.TrimSpace(out) |
|  |  |
|  | // not long running |
|  | out, \_ = dockerCmd(c, "run", "-d", "busybox", "true") |
|  | thirdID := strings.TrimSpace(out) |
|  |  |
|  | out = runSleepingContainer(c, "-d") |
|  | fourthID := strings.TrimSpace(out) |
|  |  |
|  | // make sure the second is running |
|  | assert.Assert(c, waitRun(secondID) == nil) |
|  |  |
|  | // make sure third one is not running |
|  | dockerCmd(c, "wait", thirdID) |
|  |  |
|  | // make sure the forth is running |
|  | assert.Assert(c, waitRun(fourthID) == nil) |
|  |  |
|  | // all |
|  | out, \_ = dockerCmd(c, "ps", "-a") |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), []string{fourthID, thirdID, secondID, firstID}), true, fmt.Sprintf("ALL: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | // running |
|  | out, \_ = dockerCmd(c, "ps") |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), []string{fourthID, secondID, firstID}), true, fmt.Sprintf("RUNNING: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | // limit |
|  | out, \_ = dockerCmd(c, "ps", "-n=2", "-a") |
|  | expected := []string{fourthID, thirdID} |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), expected), true, fmt.Sprintf("LIMIT & ALL: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | out, \_ = dockerCmd(c, "ps", "-n=2") |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), expected), true, fmt.Sprintf("LIMIT: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | // filter since |
|  | out, \_ = dockerCmd(c, "ps", "-f", "since="+firstID, "-a") |
|  | expected = []string{fourthID, thirdID, secondID} |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), expected), true, fmt.Sprintf("SINCE filter & ALL: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | out, \_ = dockerCmd(c, "ps", "-f", "since="+firstID) |
|  | expected = []string{fourthID, secondID} |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), expected), true, fmt.Sprintf("SINCE filter: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | out, \_ = dockerCmd(c, "ps", "-f", "since="+thirdID) |
|  | expected = []string{fourthID} |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), expected), true, fmt.Sprintf("SINCE filter: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | // filter before |
|  | out, \_ = dockerCmd(c, "ps", "-f", "before="+fourthID, "-a") |
|  | expected = []string{thirdID, secondID, firstID} |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), expected), true, fmt.Sprintf("BEFORE filter & ALL: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | out, \_ = dockerCmd(c, "ps", "-f", "before="+fourthID) |
|  | expected = []string{secondID, firstID} |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), expected), true, fmt.Sprintf("BEFORE filter: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | out, \_ = dockerCmd(c, "ps", "-f", "before="+thirdID) |
|  | expected = []string{secondID, firstID} |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), expected), true, fmt.Sprintf("SINCE filter: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | // filter since & before |
|  | out, \_ = dockerCmd(c, "ps", "-f", "since="+firstID, "-f", "before="+fourthID, "-a") |
|  | expected = []string{thirdID, secondID} |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), expected), true, fmt.Sprintf("SINCE filter, BEFORE filter & ALL: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | out, \_ = dockerCmd(c, "ps", "-f", "since="+firstID, "-f", "before="+fourthID) |
|  | expected = []string{secondID} |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), expected), true, fmt.Sprintf("SINCE filter, BEFORE filter: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | // filter since & limit |
|  | out, \_ = dockerCmd(c, "ps", "-f", "since="+firstID, "-n=2", "-a") |
|  | expected = []string{fourthID, thirdID} |
|  |  |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), expected), true, fmt.Sprintf("SINCE filter, LIMIT & ALL: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | out, \_ = dockerCmd(c, "ps", "-f", "since="+firstID, "-n=2") |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), expected), true, fmt.Sprintf("SINCE filter, LIMIT: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | // filter before & limit |
|  | out, \_ = dockerCmd(c, "ps", "-f", "before="+fourthID, "-n=1", "-a") |
|  | expected = []string{thirdID} |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), expected), true, fmt.Sprintf("BEFORE filter, LIMIT & ALL: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | out, \_ = dockerCmd(c, "ps", "-f", "before="+fourthID, "-n=1") |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), expected), true, fmt.Sprintf("BEFORE filter, LIMIT: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | // filter since & filter before & limit |
|  | out, \_ = dockerCmd(c, "ps", "-f", "since="+firstID, "-f", "before="+fourthID, "-n=1", "-a") |
|  | expected = []string{thirdID} |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), expected), true, fmt.Sprintf("SINCE filter, BEFORE filter, LIMIT & ALL: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | out, \_ = dockerCmd(c, "ps", "-f", "since="+firstID, "-f", "before="+fourthID, "-n=1") |
|  | assert.Equal(c, assertContainerList(RemoveOutputForExistingElements(out, existingContainers), expected), true, fmt.Sprintf("SINCE filter, BEFORE filter, LIMIT: Container list is not in the correct order: \n%s", out)) |
|  |  |
|  | } |
|  |  |
|  | func assertContainerList(out string, expected []string) bool { |
|  | lines := strings.Split(strings.Trim(out, "\n "), "\n") |
|  |  |
|  | if len(lines)-1 != len(expected) { |
|  | return false |
|  | } |
|  |  |
|  | containerIDIndex := strings.Index(lines[0], "CONTAINER ID") |
|  | for i := 0; i < len(expected); i++ { |
|  | foundID := lines[i+1][containerIDIndex : containerIDIndex+12] |
|  | if foundID != expected[i][:12] { |
|  | return false |
|  | } |
|  | } |
|  |  |
|  | return true |
|  | } |
|  |  |
|  | func (s \*DockerSuite) TestPsListContainersSize(c \*testing.T) { |
|  | // Problematic on Windows as it doesn't report the size correctly @swernli |
|  | testRequires(c, DaemonIsLinux) |
|  | dockerCmd(c, "run", "-d", "busybox") |
|  |  |
|  | baseOut, \_ := dockerCmd(c, "ps", "-s", "-n=1") |
|  | baseLines := strings.Split(strings.Trim(baseOut, "\n "), "\n") |
|  | baseSizeIndex := strings.Index(baseLines[0], "SIZE") |
|  | baseFoundsize := baseLines[1][baseSizeIndex:] |
|  | baseBytes, err := strconv.Atoi(strings.Split(baseFoundsize, "B")[0]) |
|  | assert.NilError(c, err) |
|  |  |
|  | name := "test\_size" |
|  | dockerCmd(c, "run", "--name", name, "busybox", "sh", "-c", "echo 1 > test") |
|  | id := getIDByName(c, name) |
|  |  |
|  | var result \*icmd.Result |
|  |  |
|  | wait := make(chan struct{}) |
|  | go func() { |
|  | result = icmd.RunCommand(dockerBinary, "ps", "-s", "-n=1") |
|  | close(wait) |
|  | }() |
|  | select { |
|  | case <-wait: |
|  | case <-time.After(3 \* time.Second): |
|  | c.Fatalf("Calling \"docker ps -s\" timed out!") |
|  | } |
|  | result.Assert(c, icmd.Success) |
|  | lines := strings.Split(strings.Trim(result.Combined(), "\n "), "\n") |
|  | assert.Equal(c, len(lines), 2, "Expected 2 lines for 'ps -s -n=1' output, got %d", len(lines)) |
|  | sizeIndex := strings.Index(lines[0], "SIZE") |
|  | idIndex := strings.Index(lines[0], "CONTAINER ID") |
|  | foundID := lines[1][idIndex : idIndex+12] |
|  | assert.Equal(c, foundID, id[:12], fmt.Sprintf("Expected id %s, got %s", id[:12], foundID)) |
|  | expectedSize := fmt.Sprintf("%dB", 2+baseBytes) |
|  | foundSize := lines[1][sizeIndex:] |
|  | assert.Assert(c, strings.Contains(foundSize, expectedSize), "Expected size %q, got %q", expectedSize, foundSize) |
|  | } |
|  |  |
|  | func (s \*DockerSuite) TestPsListContainersFilterStatus(c \*testing.T) { |
|  | existingContainers := ExistingContainerIDs(c) |
|  |  |
|  | // start exited container |
|  | out := cli.DockerCmd(c, "run", "-d", "busybox").Combined() |
|  | firstID := strings.TrimSpace(out) |
|  |  |
|  | // make sure the exited container is not running |
|  | cli.DockerCmd(c, "wait", firstID) |
|  |  |
|  | // start running container |
|  | out = cli.DockerCmd(c, "run", "-itd", "busybox").Combined() |
|  | secondID := strings.TrimSpace(out) |
|  |  |
|  | // filter containers by exited |
|  | out = cli.DockerCmd(c, "ps", "--no-trunc", "-q", "--filter=status=exited").Combined() |
|  | containerOut := strings.TrimSpace(out) |
|  | assert.Equal(c, RemoveOutputForExistingElements(containerOut, existingContainers), firstID) |
|  |  |
|  | out = cli.DockerCmd(c, "ps", "-a", "--no-trunc", "-q", "--filter=status=running").Combined() |
|  | containerOut = strings.TrimSpace(out) |
|  | assert.Equal(c, RemoveOutputForExistingElements(containerOut, existingContainers), secondID) |
|  |  |
|  | result := cli.Docker(cli.Args("ps", "-a", "-q", "--filter=status=rubbish"), cli.WithTimeout(time.Second\*60)) |
|  | err := "Invalid filter 'status=rubbish'" |
|  | if versions.LessThan(testEnv.DaemonAPIVersion(), "1.32") { |
|  | err = "Unrecognised filter value for status: rubbish" |
|  | } |
|  | result.Assert(c, icmd.Expected{ |
|  | ExitCode: 1, |
|  | Err: err, |
|  | }) |
|  | // Windows doesn't support pausing of containers |
|  | if testEnv.OSType != "windows" { |
|  | // pause running container |
|  | out = cli.DockerCmd(c, "run", "-itd", "busybox").Combined() |
|  | pausedID := strings.TrimSpace(out) |
|  | cli.DockerCmd(c, "pause", pausedID) |
|  | // make sure the container is unpaused to let the daemon stop it properly |
|  | defer func() { cli.DockerCmd(c, "unpause", pausedID) }() |
|  |  |
|  | out = cli.DockerCmd(c, "ps", "--no-trunc", "-q", "--filter=status=paused").Combined() |
|  | containerOut = strings.TrimSpace(out) |
|  | assert.Equal(c, RemoveOutputForExistingElements(containerOut, existingContainers), pausedID) |
|  | } |
|  | } |
|  |  |
|  | func (s \*DockerSuite) TestPsListContainersFilterHealth(c \*testing.T) { |
|  | existingContainers := ExistingContainerIDs(c) |
|  | // Test legacy no health check |
|  | out := runSleepingContainer(c, "--name=none\_legacy") |
|  | containerID := strings.TrimSpace(out) |
|  |  |
|  | cli.WaitRun(c, containerID) |
|  |  |
|  | out = cli.DockerCmd(c, "ps", "-q", "-l", "--no-trunc", "--filter=health=none").Combined() |
|  | containerOut := strings.TrimSpace(out) |
|  | assert.Equal(c, containerOut, containerID, fmt.Sprintf("Expected id %s, got %s for legacy none filter, output: %q", containerID, containerOut, out)) |
|  |  |
|  | // Test no health check specified explicitly |
|  | out = runSleepingContainer(c, "--name=none", "--no-healthcheck") |
|  | containerID = strings.TrimSpace(out) |
|  |  |
|  | cli.WaitRun(c, containerID) |
|  |  |
|  | out = cli.DockerCmd(c, "ps", "-q", "-l", "--no-trunc", "--filter=health=none").Combined() |
|  | containerOut = strings.TrimSpace(out) |
|  | assert.Equal(c, containerOut, containerID, fmt.Sprintf("Expected id %s, got %s for none filter, output: %q", containerID, containerOut, out)) |
|  |  |
|  | // Test failing health check |
|  | out = runSleepingContainer(c, "--name=failing\_container", "--health-cmd=exit 1", "--health-interval=1s") |
|  | containerID = strings.TrimSpace(out) |
|  |  |
|  | waitForHealthStatus(c, "failing\_container", "starting", "unhealthy") |
|  |  |
|  | out = cli.DockerCmd(c, "ps", "-q", "--no-trunc", "--filter=health=unhealthy").Combined() |
|  | containerOut = strings.TrimSpace(out) |
|  | assert.Equal(c, containerOut, containerID, fmt.Sprintf("Expected containerID %s, got %s for unhealthy filter, output: %q", containerID, containerOut, out)) |
|  |  |
|  | // Check passing healthcheck |
|  | out = runSleepingContainer(c, "--name=passing\_container", "--health-cmd=exit 0", "--health-interval=1s") |
|  | containerID = strings.TrimSpace(out) |
|  |  |
|  | waitForHealthStatus(c, "passing\_container", "starting", "healthy") |
|  |  |
|  | out = cli.DockerCmd(c, "ps", "-q", "--no-trunc", "--filter=health=healthy").Combined() |
|  | containerOut = strings.TrimSpace(RemoveOutputForExistingElements(out, existingContainers)) |
|  | assert.Equal(c, containerOut, containerID, fmt.Sprintf("Expected containerID %s, got %s for healthy filter, output: %q", containerID, containerOut, out)) |
|  | } |
|  |  |
|  | func (s \*DockerSuite) TestPsListContainersFilterID(c \*testing.T) { |
|  | // start container |
|  | out, \_ := dockerCmd(c, "run", "-d", "busybox") |
|  | firstID := strings.TrimSpace(out) |
|  |  |
|  | // start another container |
|  | runSleepingContainer(c) |
|  |  |
|  | // filter containers by id |
|  | out, \_ = dockerCmd(c, "ps", "-a", "-q", "--filter=id="+firstID) |
|  | containerOut := strings.TrimSpace(out) |
|  | assert.Equal(c, containerOut, firstID[:12], fmt.Sprintf("Expected id %s, got %s for exited filter, output: %q", firstID[:12], containerOut, out)) |
|  | } |
|  |  |
|  | func (s \*DockerSuite) TestPsListContainersFilterName(c \*testing.T) { |
|  | // start container |
|  | dockerCmd(c, "run", "--name=a\_name\_to\_match", "busybox") |
|  | id := getIDByName(c, "a\_name\_to\_match") |
|  |  |
|  | // start another container |
|  | runSleepingContainer(c, "--name=b\_name\_to\_match") |
|  |  |
|  | // filter containers by name |
|  | out, \_ := dockerCmd(c, "ps", "-a", "-q", "--filter=name=a\_name\_to\_match") |
|  | containerOut := strings.TrimSpace(out) |
|  | assert.Equal(c, containerOut, id[:12], fmt.Sprintf("Expected id %s, got %s for exited filter, output: %q", id[:12], containerOut, out)) |
|  | } |
|  |  |
|  | // Test for the ancestor filter for ps. |
|  | // There is also the same test but with image:tag@digest in docker\_cli\_by\_digest\_test.go |
|  | // |
|  | // What the test setups : |
|  | // - Create 2 image based on busybox using the same repository but different tags |
|  | // - Create an image based on the previous image (images\_ps\_filter\_test2) |
|  | // - Run containers for each of those image (busybox, images\_ps\_filter\_test1, images\_ps\_filter\_test2) |
|  | // - Filter them out :P |
|  | func (s \*DockerSuite) TestPsListContainersFilterAncestorImage(c \*testing.T) { |
|  | existingContainers := ExistingContainerIDs(c) |
|  |  |
|  | // Build images |
|  | imageName1 := "images\_ps\_filter\_test1" |
|  | buildImageSuccessfully(c, imageName1, build.WithDockerfile(`FROM busybox |
|  | LABEL match me 1`)) |
|  | imageID1 := getIDByName(c, imageName1) |
|  |  |
|  | imageName1Tagged := "images\_ps\_filter\_test1:tag" |
|  | buildImageSuccessfully(c, imageName1Tagged, build.WithDockerfile(`FROM busybox |
|  | LABEL match me 1 tagged`)) |
|  | imageID1Tagged := getIDByName(c, imageName1Tagged) |
|  |  |
|  | imageName2 := "images\_ps\_filter\_test2" |
|  | buildImageSuccessfully(c, imageName2, build.WithDockerfile(fmt.Sprintf(`FROM %s |
|  | LABEL match me 2`, imageName1))) |
|  | imageID2 := getIDByName(c, imageName2) |
|  |  |
|  | // start containers |
|  | dockerCmd(c, "run", "--name=first", "busybox", "echo", "hello") |
|  | firstID := getIDByName(c, "first") |
|  |  |
|  | // start another container |
|  | dockerCmd(c, "run", "--name=second", "busybox", "echo", "hello") |
|  | secondID := getIDByName(c, "second") |
|  |  |
|  | // start third container |
|  | dockerCmd(c, "run", "--name=third", imageName1, "echo", "hello") |
|  | thirdID := getIDByName(c, "third") |
|  |  |
|  | // start fourth container |
|  | dockerCmd(c, "run", "--name=fourth", imageName1Tagged, "echo", "hello") |
|  | fourthID := getIDByName(c, "fourth") |
|  |  |
|  | // start fifth container |
|  | dockerCmd(c, "run", "--name=fifth", imageName2, "echo", "hello") |
|  | fifthID := getIDByName(c, "fifth") |
|  |  |
|  | var filterTestSuite = []struct { |
|  | filterName string |
|  | expectedIDs []string |
|  | }{ |
|  | // non existent stuff |
|  | {"nonexistent", []string{}}, |
|  | {"nonexistent:tag", []string{}}, |
|  | // image |
|  | {"busybox", []string{firstID, secondID, thirdID, fourthID, fifthID}}, |
|  | {imageName1, []string{thirdID, fifthID}}, |
|  | {imageName2, []string{fifthID}}, |
|  | // image:tag |
|  | {fmt.Sprintf("%s:latest", imageName1), []string{thirdID, fifthID}}, |
|  | {imageName1Tagged, []string{fourthID}}, |
|  | // short-id |
|  | {stringid.TruncateID(imageID1), []string{thirdID, fifthID}}, |
|  | {stringid.TruncateID(imageID2), []string{fifthID}}, |
|  | // full-id |
|  | {imageID1, []string{thirdID, fifthID}}, |
|  | {imageID1Tagged, []string{fourthID}}, |
|  | {imageID2, []string{fifthID}}, |
|  | } |
|  |  |
|  | var out string |
|  | for \_, filter := range filterTestSuite { |
|  | out, \_ = dockerCmd(c, "ps", "-a", "-q", "--no-trunc", "--filter=ancestor="+filter.filterName) |
|  | checkPsAncestorFilterOutput(c, RemoveOutputForExistingElements(out, existingContainers), filter.filterName, filter.expectedIDs) |
|  | } |
|  |  |
|  | // Multiple ancestor filter |
|  | out, \_ = dockerCmd(c, "ps", "-a", "-q", "--no-trunc", "--filter=ancestor="+imageName2, "--filter=ancestor="+imageName1Tagged) |
|  | checkPsAncestorFilterOutput(c, RemoveOutputForExistingElements(out, existingContainers), imageName2+","+imageName1Tagged, []string{fourthID, fifthID}) |
|  | } |
|  |  |
|  | func checkPsAncestorFilterOutput(c \*testing.T, out string, filterName string, expectedIDs []string) { |
|  | var actualIDs []string |
|  | if out != "" { |
|  | actualIDs = strings.Split(out[:len(out)-1], "\n") |
|  | } |
|  | sort.Strings(actualIDs) |
|  | sort.Strings(expectedIDs) |
|  |  |
|  | assert.Equal(c, len(actualIDs), len(expectedIDs), fmt.Sprintf("Expected filtered container(s) for %s ancestor filter to be %v:%v, got %v:%v", filterName, len(expectedIDs), expectedIDs, len(actualIDs), actualIDs)) |
|  | if len(expectedIDs) > 0 { |
|  | same := true |
|  | for i := range expectedIDs { |
|  | if actualIDs[i] != expectedIDs[i] { |
|  | c.Logf("%s, %s", actualIDs[i], expectedIDs[i]) |
|  | same = false |
|  | break |
|  | } |
|  | } |
|  | assert.Equal(c, same, true, fmt.Sprintf("Expected filtered container(s) for %s ancestor filter to be %v, got %v", filterName, expectedIDs, actualIDs)) |
|  | } |
|  | } |
|  |  |
|  | func (s \*DockerSuite) TestPsListContainersFilterLabel(c \*testing.T) { |
|  | // start container |
|  | dockerCmd(c, "run", "--name=first", "-l", "match=me", "-l", "second=tag", "busybox") |
|  | firstID := getIDByName(c, "first") |
|  |  |
|  | // start another container |
|  | dockerCmd(c, "run", "--name=second", "-l", "match=me too", "busybox") |
|  | secondID := getIDByName(c, "second") |
|  |  |
|  | // start third container |
|  | dockerCmd(c, "run", "--name=third", "-l", "nomatch=me", "busybox") |
|  | thirdID := getIDByName(c, "third") |
|  |  |
|  | // filter containers by exact match |
|  | out, \_ := dockerCmd(c, "ps", "-a", "-q", "--no-trunc", "--filter=label=match=me") |
|  | containerOut := strings.TrimSpace(out) |
|  | assert.Equal(c, containerOut, firstID, fmt.Sprintf("Expected id %s, got %s for exited filter, output: %q", firstID, containerOut, out)) |
|  |  |
|  | // filter containers by two labels |
|  | out, \_ = dockerCmd(c, "ps", "-a", "-q", "--no-trunc", "--filter=label=match=me", "--filter=label=second=tag") |
|  | containerOut = strings.TrimSpace(out) |
|  | assert.Equal(c, containerOut, firstID, fmt.Sprintf("Expected id %s, got %s for exited filter, output: %q", firstID, containerOut, out)) |
|  |  |
|  | // filter containers by two labels, but expect not found because of AND behavior |
|  | out, \_ = dockerCmd(c, "ps", "-a", "-q", "--no-trunc", "--filter=label=match=me", "--filter=label=second=tag-no") |
|  | containerOut = strings.TrimSpace(out) |
|  | assert.Equal(c, containerOut, "", fmt.Sprintf("Expected nothing, got %s for exited filter, output: %q", containerOut, out)) |
|  |  |
|  | // filter containers by exact key |
|  | out, \_ = dockerCmd(c, "ps", "-a", "-q", "--no-trunc", "--filter=label=match") |
|  | containerOut = strings.TrimSpace(out) |
|  | assert.Assert(c, strings.Contains(containerOut, firstID)) |
|  | assert.Assert(c, strings.Contains(containerOut, secondID)) |
|  | assert.Assert(c, !strings.Contains(containerOut, thirdID)) |
|  | } |
|  |  |
|  | func (s \*DockerSuite) TestPsListContainersFilterExited(c \*testing.T) { |
|  | // TODO Flaky on Windows CI [both RS1 and RS5] |
|  | // On slower machines the container may not have exited |
|  | // yet when we filter below by exit status/exit value. |
|  | skip.If(c, DaemonIsWindows(), "FLAKY on Windows, see #20819") |
|  | runSleepingContainer(c, "--name=sleep") |
|  |  |
|  | firstZero, \_ := dockerCmd(c, "run", "-d", "busybox", "true") |
|  | secondZero, \_ := dockerCmd(c, "run", "-d", "busybox", "true") |
|  |  |
|  | out, \_, err := dockerCmdWithError("run", "--name", "nonzero1", "busybox", "false") |
|  | assert.Assert(c, err != nil, "Should fail. out: %s", out) |
|  | firstNonZero := getIDByName(c, "nonzero1") |
|  |  |
|  | out, \_, err = dockerCmdWithError("run", "--name", "nonzero2", "busybox", "false") |
|  | assert.Assert(c, err != nil, "Should fail. out: %s", out) |
|  | secondNonZero := getIDByName(c, "nonzero2") |
|  |  |
|  | // filter containers by exited=0 |
|  | out, \_ = dockerCmd(c, "ps", "-a", "-q", "--no-trunc", "--filter=exited=0") |
|  | assert.Assert(c, strings.Contains(out, strings.TrimSpace(firstZero))) |
|  | assert.Assert(c, strings.Contains(out, strings.TrimSpace(secondZero))) |
|  | assert.Assert(c, !strings.Contains(out, strings.TrimSpace(firstNonZero))) |
|  | assert.Assert(c, !strings.Contains(out, strings.TrimSpace(secondNonZero))) |
|  | out, \_ = dockerCmd(c, "ps", "-a", "-q", "--no-trunc", "--filter=exited=1") |
|  | assert.Assert(c, strings.Contains(out, strings.TrimSpace(firstNonZero))) |
|  | assert.Assert(c, strings.Contains(out, strings.TrimSpace(secondNonZero))) |
|  | assert.Assert(c, !strings.Contains(out, strings.TrimSpace(firstZero))) |
|  | assert.Assert(c, !strings.Contains(out, strings.TrimSpace(secondZero))) |
|  | } |
|  |  |
|  | func (s \*DockerSuite) TestPsRightTagName(c \*testing.T) { |
|  | // TODO Investigate further why this fails on Windows to Windows CI |
|  | testRequires(c, DaemonIsLinux) |
|  |  |
|  | existingContainers := ExistingContainerNames(c) |
|  |  |
|  | tag := "asybox:shmatest" |
|  | dockerCmd(c, "tag", "busybox", tag) |
|  |  |
|  | var id1 string |
|  | out := runSleepingContainer(c) |
|  | id1 = strings.TrimSpace(out) |
|  |  |
|  | var id2 string |
|  | out = runSleepingContainerInImage(c, tag) |
|  | id2 = strings.TrimSpace(out) |
|  |  |
|  | var imageID string |
|  | out = inspectField(c, "busybox", "Id") |
|  | imageID = strings.TrimSpace(out) |
|  |  |
|  | var id3 string |
|  | out = runSleepingContainerInImage(c, imageID) |
|  | id3 = strings.TrimSpace(out) |
|  |  |
|  | out, \_ = dockerCmd(c, "ps", "--no-trunc") |
|  | lines := strings.Split(strings.TrimSpace(out), "\n") |
|  | lines = RemoveLinesForExistingElements(lines, existingContainers) |
|  | // skip header |
|  | lines = lines[1:] |
|  | assert.Equal(c, len(lines), 3, "There should be 3 running container, got %d", len(lines)) |
|  | for \_, line := range lines { |
|  | f := strings.Fields(line) |
|  | switch f[0] { |
|  | case id1: |
|  | assert.Equal(c, f[1], "busybox", fmt.Sprintf("Expected %s tag for id %s, got %s", "busybox", id1, f[1])) |
|  | case id2: |
|  | assert.Equal(c, f[1], tag, fmt.Sprintf("Expected %s tag for id %s, got %s", tag, id2, f[1])) |
|  | case id3: |
|  | assert.Equal(c, f[1], imageID, fmt.Sprintf("Expected %s imageID for id %s, got %s", tag, id3, f[1])) |
|  | default: |
|  | c.Fatalf("Unexpected id %s, expected %s and %s and %s", f[0], id1, id2, id3) |
|  | } |
|  | } |
|  | } |
|  |  |
|  | func (s \*DockerSuite) TestPsListContainersFilterCreated(c \*testing.T) { |
|  | // create a container |
|  | out, \_ := dockerCmd(c, "create", "busybox") |
|  | cID := strings.TrimSpace(out) |
|  | shortCID := cID[:12] |
|  |  |
|  | // Make sure it DOESN'T show up w/o a '-a' for normal 'ps' |
|  | out, \_ = dockerCmd(c, "ps", "-q") |
|  | assert.Assert(c, !strings.Contains(out, shortCID), "Should have not seen '%s' in ps output:\n%s", shortCID, out) |
|  | // Make sure it DOES show up as 'Created' for 'ps -a' |
|  | out, \_ = dockerCmd(c, "ps", "-a") |
|  |  |
|  | hits := 0 |
|  | for \_, line := range strings.Split(out, "\n") { |
|  | if !strings.Contains(line, shortCID) { |
|  | continue |
|  | } |
|  | hits++ |
|  | assert.Assert(c, strings.Contains(line, "Created"), "Missing 'Created' on '%s'", line) |
|  | } |
|  |  |
|  | assert.Equal(c, hits, 1, fmt.Sprintf("Should have seen '%s' in ps -a output once:%d\n%s", shortCID, hits, out)) |
|  |  |
|  | // filter containers by 'create' - note, no -a needed |
|  | out, \_ = dockerCmd(c, "ps", "-q", "-f", "status=created") |
|  | containerOut := strings.TrimSpace(out) |
|  | assert.Assert(c, strings.HasPrefix(cID, containerOut)) |
|  | } |
|  |  |
|  | // Test for GitHub issue #12595 |
|  | func (s \*DockerSuite) TestPsImageIDAfterUpdate(c \*testing.T) { |
|  | // TODO: Investigate why this fails on Windows to Windows CI further. |
|  | testRequires(c, DaemonIsLinux) |
|  | originalImageName := "busybox:TestPsImageIDAfterUpdate-original" |
|  | updatedImageName := "busybox:TestPsImageIDAfterUpdate-updated" |
|  |  |
|  | existingContainers := ExistingContainerIDs(c) |
|  |  |
|  | icmd.RunCommand(dockerBinary, "tag", "busybox:latest", originalImageName).Assert(c, icmd.Success) |
|  |  |
|  | originalImageID := getIDByName(c, originalImageName) |
|  |  |
|  | result := icmd.RunCommand(dockerBinary, append([]string{"run", "-d", originalImageName}, sleepCommandForDaemonPlatform()...)...) |
|  | result.Assert(c, icmd.Success) |
|  | containerID := strings.TrimSpace(result.Combined()) |
|  |  |
|  | result = icmd.RunCommand(dockerBinary, "ps", "--no-trunc") |
|  | result.Assert(c, icmd.Success) |
|  |  |
|  | lines := strings.Split(strings.TrimSpace(result.Combined()), "\n") |
|  | lines = RemoveLinesForExistingElements(lines, existingContainers) |
|  | // skip header |
|  | lines = lines[1:] |
|  | assert.Equal(c, len(lines), 1) |
|  |  |
|  | for \_, line := range lines { |
|  | f := strings.Fields(line) |
|  | assert.Equal(c, f[1], originalImageName) |
|  | } |
|  |  |
|  | icmd.RunCommand(dockerBinary, "commit", containerID, updatedImageName).Assert(c, icmd.Success) |
|  | icmd.RunCommand(dockerBinary, "tag", updatedImageName, originalImageName).Assert(c, icmd.Success) |
|  |  |
|  | result = icmd.RunCommand(dockerBinary, "ps", "--no-trunc") |
|  | result.Assert(c, icmd.Success) |
|  |  |
|  | lines = strings.Split(strings.TrimSpace(result.Combined()), "\n") |
|  | lines = RemoveLinesForExistingElements(lines, existingContainers) |
|  | // skip header |
|  | lines = lines[1:] |
|  | assert.Equal(c, len(lines), 1) |
|  |  |
|  | for \_, line := range lines { |
|  | f := strings.Fields(line) |
|  | assert.Equal(c, f[1], originalImageID) |
|  | } |
|  |  |
|  | } |
|  |  |
|  | func (s \*DockerSuite) TestPsNotShowPortsOfStoppedContainer(c \*testing.T) { |
|  | testRequires(c, DaemonIsLinux) |
|  | dockerCmd(c, "run", "--name=foo", "-d", "-p", "5000:5000", "busybox", "top") |
|  | assert.Assert(c, waitRun("foo") == nil) |
|  | out, \_ := dockerCmd(c, "ps") |
|  | lines := strings.Split(strings.TrimSpace(out), "\n") |
|  | expected := "0.0.0.0:5000->5000/tcp" |
|  | fields := strings.Fields(lines[1]) |
|  | assert.Equal(c, fields[len(fields)-2], expected, fmt.Sprintf("Expected: %v, got: %v", expected, fields[len(fields)-2])) |
|  |  |
|  | dockerCmd(c, "kill", "foo") |
|  | dockerCmd(c, "wait", "foo") |
|  | out, \_ = dockerCmd(c, "ps", "-l") |
|  | lines = strings.Split(strings.TrimSpace(out), "\n") |
|  | fields = strings.Fields(lines[1]) |
|  | assert.Assert(c, fields[len(fields)-2] != expected, "Should not got %v", expected) |
|  | } |
|  |  |
|  | func (s \*DockerSuite) TestPsShowMounts(c \*testing.T) { |
|  | existingContainers := ExistingContainerNames(c) |
|  |  |
|  | prefix, slash := getPrefixAndSlashFromDaemonPlatform() |
|  |  |
|  | mp := prefix + slash + "test" |
|  |  |
|  | dockerCmd(c, "volume", "create", "ps-volume-test") |
|  | // volume mount containers |
|  | runSleepingContainer(c, "--name=volume-test-1", "--volume", "ps-volume-test:"+mp) |
|  | assert.Assert(c, waitRun("volume-test-1") == nil) |
|  | runSleepingContainer(c, "--name=volume-test-2", "--volume", mp) |
|  | assert.Assert(c, waitRun("volume-test-2") == nil) |
|  | // bind mount container |
|  | var bindMountSource string |
|  | var bindMountDestination string |
|  | if DaemonIsWindows() { |
|  | bindMountSource = "c:\\" |
|  | bindMountDestination = "c:\\t" |
|  | } else { |
|  | bindMountSource = "/tmp" |
|  | bindMountDestination = "/t" |
|  | } |
|  | runSleepingContainer(c, "--name=bind-mount-test", "-v", bindMountSource+":"+bindMountDestination) |
|  | assert.Assert(c, waitRun("bind-mount-test") == nil) |
|  |  |
|  | out, \_ := dockerCmd(c, "ps", "--format", "{{.Names}} {{.Mounts}}") |
|  |  |
|  | lines := strings.Split(strings.TrimSpace(out), "\n") |
|  | lines = RemoveLinesForExistingElements(lines, existingContainers) |
|  | assert.Equal(c, len(lines), 3) |
|  |  |
|  | fields := strings.Fields(lines[0]) |
|  | assert.Equal(c, len(fields), 2) |
|  | assert.Equal(c, fields[0], "bind-mount-test") |
|  | assert.Equal(c, fields[1], bindMountSource) |
|  |  |
|  | fields = strings.Fields(lines[1]) |
|  | assert.Equal(c, len(fields), 2) |
|  |  |
|  | anonymousVolumeID := fields[1] |
|  |  |
|  | fields = strings.Fields(lines[2]) |
|  | assert.Equal(c, fields[1], "ps-volume-test") |
|  |  |
|  | // filter by volume name |
|  | out, \_ = dockerCmd(c, "ps", "--format", "{{.Names}} {{.Mounts}}", "--filter", "volume=ps-volume-test") |
|  |  |
|  | lines = strings.Split(strings.TrimSpace(out), "\n") |
|  | lines = RemoveLinesForExistingElements(lines, existingContainers) |
|  | assert.Equal(c, len(lines), 1) |
|  |  |
|  | fields = strings.Fields(lines[0]) |
|  | assert.Equal(c, fields[1], "ps-volume-test") |
|  |  |
|  | // empty results filtering by unknown volume |
|  | out, \_ = dockerCmd(c, "ps", "--format", "{{.Names}} {{.Mounts}}", "--filter", "volume=this-volume-should-not-exist") |
|  | assert.Equal(c, len(strings.TrimSpace(out)), 0) |
|  |  |
|  | // filter by mount destination |
|  | out, \_ = dockerCmd(c, "ps", "--format", "{{.Names}} {{.Mounts}}", "--filter", "volume="+mp) |
|  |  |
|  | lines = strings.Split(strings.TrimSpace(out), "\n") |
|  | lines = RemoveLinesForExistingElements(lines, existingContainers) |
|  | assert.Equal(c, len(lines), 2) |
|  |  |
|  | fields = strings.Fields(lines[0]) |
|  | assert.Equal(c, fields[1], anonymousVolumeID) |
|  | fields = strings.Fields(lines[1]) |
|  | assert.Equal(c, fields[1], "ps-volume-test") |
|  |  |
|  | // filter by bind mount source |
|  | out, \_ = dockerCmd(c, "ps", "--format", "{{.Names}} {{.Mounts}}", "--filter", "volume="+bindMountSource) |
|  |  |
|  | lines = strings.Split(strings.TrimSpace(out), "\n") |
|  | lines = RemoveLinesForExistingElements(lines, existingContainers) |
|  | assert.Equal(c, len(lines), 1) |
|  |  |
|  | fields = strings.Fields(lines[0]) |
|  | assert.Equal(c, len(fields), 2) |
|  | assert.Equal(c, fields[0], "bind-mount-test") |
|  | assert.Equal(c, fields[1], bindMountSource) |
|  |  |
|  | // filter by bind mount destination |
|  | out, \_ = dockerCmd(c, "ps", "--format", "{{.Names}} {{.Mounts}}", "--filter", "volume="+bindMountDestination) |
|  |  |
|  | lines = strings.Split(strings.TrimSpace(out), "\n") |
|  | lines = RemoveLinesForExistingElements(lines, existingContainers) |
|  | assert.Equal(c, len(lines), 1) |
|  |  |
|  | fields = strings.Fields(lines[0]) |
|  | assert.Equal(c, len(fields), 2) |
|  | assert.Equal(c, fields[0], "bind-mount-test") |
|  | assert.Equal(c, fields[1], bindMountSource) |
|  |  |
|  | // empty results filtering by unknown mount point |
|  | out, \_ = dockerCmd(c, "ps", "--format", "{{.Names}} {{.Mounts}}", "--filter", "volume="+prefix+slash+"this-path-was-never-mounted") |
|  | assert.Equal(c, len(strings.TrimSpace(out)), 0) |
|  | } |
|  |  |
|  | func (s \*DockerSuite) TestPsListContainersFilterNetwork(c \*testing.T) { |
|  | existing := ExistingContainerIDs(c) |
|  |  |
|  | // TODO default network on Windows is not called "bridge", and creating a |
|  | // custom network fails on Windows fails with "Error response from daemon: plugin not found") |
|  | testRequires(c, DaemonIsLinux) |
|  |  |
|  | // create some containers |
|  | runSleepingContainer(c, "--net=bridge", "--name=onbridgenetwork") |
|  | runSleepingContainer(c, "--net=none", "--name=onnonenetwork") |
|  |  |
|  | // Filter docker ps on non existing network |
|  | out, \_ := dockerCmd(c, "ps", "--filter", "network=doesnotexist") |
|  | containerOut := strings.TrimSpace(out) |
|  | lines := strings.Split(containerOut, "\n") |
|  |  |
|  | // skip header |
|  | lines = lines[1:] |
|  |  |
|  | // ps output should have no containers |
|  | assert.Equal(c, len(RemoveLinesForExistingElements(lines, existing)), 0) |
|  |  |
|  | // Filter docker ps on network bridge |
|  | out, \_ = dockerCmd(c, "ps", "--filter", "network=bridge") |
|  | containerOut = strings.TrimSpace(out) |
|  |  |
|  | lines = strings.Split(containerOut, "\n") |
|  |  |
|  | // skip header |
|  | lines = lines[1:] |
|  |  |
|  | // ps output should have only one container |
|  | assert.Equal(c, len(RemoveLinesForExistingElements(lines, existing)), 1) |
|  |  |
|  | // Making sure onbridgenetwork is on the output |
|  | assert.Assert(c, strings.Contains(containerOut, "onbridgenetwork"), "Missing the container on network\n") |
|  | // Filter docker ps on networks bridge and none |
|  | out, \_ = dockerCmd(c, "ps", "--filter", "network=bridge", "--filter", "network=none") |
|  | containerOut = strings.TrimSpace(out) |
|  |  |
|  | lines = strings.Split(containerOut, "\n") |
|  |  |
|  | // skip header |
|  | lines = lines[1:] |
|  |  |
|  | // ps output should have both the containers |
|  | assert.Equal(c, len(RemoveLinesForExistingElements(lines, existing)), 2) |
|  |  |
|  | // Making sure onbridgenetwork and onnonenetwork is on the output |
|  | assert.Assert(c, strings.Contains(containerOut, "onnonenetwork"), "Missing the container on none network\n") |
|  | assert.Assert(c, strings.Contains(containerOut, "onbridgenetwork"), "Missing the container on bridge network\n") |
|  | nwID, \_ := dockerCmd(c, "network", "inspect", "--format", "{{.ID}}", "bridge") |
|  |  |
|  | // Filter by network ID |
|  | out, \_ = dockerCmd(c, "ps", "--filter", "network="+nwID) |
|  | containerOut = strings.TrimSpace(out) |
|  |  |
|  | assert.Assert(c, is.Contains(containerOut, "onbridgenetwork")) |
|  |  |
|  | // Filter by partial network ID |
|  | partialnwID := nwID[0:4] |
|  |  |
|  | out, \_ = dockerCmd(c, "ps", "--filter", "network="+partialnwID) |
|  | containerOut = strings.TrimSpace(out) |
|  |  |
|  | lines = strings.Split(containerOut, "\n") |
|  |  |
|  | // skip header |
|  | lines = lines[1:] |
|  |  |
|  | // ps output should have only one container |
|  | assert.Equal(c, len(RemoveLinesForExistingElements(lines, existing)), 1) |
|  |  |
|  | // Making sure onbridgenetwork is on the output |
|  | assert.Assert(c, strings.Contains(containerOut, "onbridgenetwork"), "Missing the container on network\n") |
|  | } |
|  |  |
|  | func (s \*DockerSuite) TestPsByOrder(c \*testing.T) { |
|  | out := runSleepingContainer(c, "--name", "xyz-abc") |
|  | container1 := strings.TrimSpace(out) |
|  |  |
|  | out = runSleepingContainer(c, "--name", "xyz-123") |
|  | container2 := strings.TrimSpace(out) |
|  |  |
|  | runSleepingContainer(c, "--name", "789-abc") |
|  | runSleepingContainer(c, "--name", "789-123") |
|  |  |
|  | // Run multiple time should have the same result |
|  | out = cli.DockerCmd(c, "ps", "--no-trunc", "-q", "-f", "name=xyz").Combined() |
|  | assert.Equal(c, strings.TrimSpace(out), fmt.Sprintf("%s\n%s", container2, container1)) |
|  |  |
|  | // Run multiple time should have the same result |
|  | out = cli.DockerCmd(c, "ps", "--no-trunc", "-q", "-f", "name=xyz").Combined() |
|  | assert.Equal(c, strings.TrimSpace(out), fmt.Sprintf("%s\n%s", container2, container1)) |
|  | } |
|  |  |
|  | func (s \*DockerSuite) TestPsListContainersFilterPorts(c \*testing.T) { |
|  | testRequires(c, DaemonIsLinux) |
|  | existingContainers := ExistingContainerIDs(c) |
|  |  |
|  | out, \_ := dockerCmd(c, "run", "-d", "--publish=80", "busybox", "top") |
|  | id1 := strings.TrimSpace(out) |
|  |  |
|  | out, \_ = dockerCmd(c, "run", "-d", "--expose=8080", "busybox", "top") |
|  | id2 := strings.TrimSpace(out) |
|  |  |
|  | out, \_ = dockerCmd(c, "ps", "--no-trunc", "-q") |
|  | assert.Assert(c, strings.Contains(strings.TrimSpace(out), id1)) |
|  | assert.Assert(c, strings.Contains(strings.TrimSpace(out), id2)) |
|  | out, \_ = dockerCmd(c, "ps", "--no-trunc", "-q", "--filter", "publish=80-8080/udp") |
|  | assert.Assert(c, strings.TrimSpace(out) != id1) |
|  | assert.Assert(c, strings.TrimSpace(out) != id2) |
|  |  |
|  | out, \_ = dockerCmd(c, "ps", "--no-trunc", "-q", "--filter", "expose=8081") |
|  | assert.Assert(c, strings.TrimSpace(out) != id1) |
|  | assert.Assert(c, strings.TrimSpace(out) != id2) |
|  |  |
|  | out, \_ = dockerCmd(c, "ps", "--no-trunc", "-q", "--filter", "publish=80-81") |
|  | assert.Equal(c, strings.TrimSpace(out), id1) |
|  | assert.Assert(c, strings.TrimSpace(out) != id2) |
|  |  |
|  | out, \_ = dockerCmd(c, "ps", "--no-trunc", "-q", "--filter", "expose=80/tcp") |
|  | assert.Equal(c, strings.TrimSpace(out), id1) |
|  | assert.Assert(c, strings.TrimSpace(out) != id2) |
|  |  |
|  | out, \_ = dockerCmd(c, "ps", "--no-trunc", "-q", "--filter", "expose=8080/tcp") |
|  | out = RemoveOutputForExistingElements(out, existingContainers) |
|  | assert.Assert(c, strings.TrimSpace(out) != id1) |
|  | assert.Equal(c, strings.TrimSpace(out), id2) |
|  | } |
|  |  |
|  | func (s \*DockerSuite) TestPsNotShowLinknamesOfDeletedContainer(c \*testing.T) { |
|  | testRequires(c, DaemonIsLinux, MinimumAPIVersion("1.31")) |
|  | existingContainers := ExistingContainerNames(c) |
|  |  |
|  | dockerCmd(c, "create", "--name=aaa", "busybox", "top") |
|  | dockerCmd(c, "create", "--name=bbb", "--link=aaa", "busybox", "top") |
|  |  |
|  | out, \_ := dockerCmd(c, "ps", "--no-trunc", "-a", "--format", "{{.Names}}") |
|  | lines := strings.Split(strings.TrimSpace(out), "\n") |
|  | lines = RemoveLinesForExistingElements(lines, existingContainers) |
|  | expected := []string{"bbb", "aaa,bbb/aaa"} |
|  | var names []string |
|  | names = append(names, lines...) |
|  | assert.Assert(c, is.DeepEqual(names, expected), "Expected array with non-truncated names: %v, got: %v", expected, names) |
|  |  |
|  | dockerCmd(c, "rm", "bbb") |
|  |  |
|  | out, \_ = dockerCmd(c, "ps", "--no-trunc", "-a", "--format", "{{.Names}}") |
|  | out = RemoveOutputForExistingElements(out, existingContainers) |
|  | assert.Equal(c, strings.TrimSpace(out), "aaa") |
|  | } |